

Best of MacTutor

The Macintosh Programming Journal



Volume 5

The Best of MacTutor

The Macintosh Programming Journal

Volume 5

Edited by
David E. Smith
and
Kirk Chase



Published by
MacTutor®

© 1990 by MacTutor®, Anaheim CA. 92807

This book was created on a Macintosh Plus Computer
using a LaserWriter Plus and Aldus Pagemaker Software.

All rights reserved. No part of this book
may be reproduced, in any form or by any means,
without permission in writing from the publisher.

Printed in the United States of America
by Webcraft Craftmanship Printing, Garden Grove, CA 92641

10 9 8 7 6 5 4 3 2

ISBN 0-9617544-4-3

*MacTutor® and the MacTutor Man Logo are registered trademarks of
the MacTutor Company.*



MacTutor®

The Macintosh Programming Journal

P.O. Box 400

or

1250 Lakeview, Suite O

Placentia, CA 92670

Anaheim, CA 92807

We would like to thank Kirk Chase and Laura Smith for putting this manuscript together. We would also like to thank all those authors who contributed their time and knowledge by writing the fine articles you see within these pages and those that did not make it.

Most importantly, we would like to thank the excellent staff at MacTutor. Just check out the staff in any MacTutor to find those people who contribute so much in putting out this journal month after month.

We regret that not all the articles of 1989 made it into this manuscript. Like our monthly publication, there were more articles than publishing space allowed. In order to have this book bound, we were forced to edit the Letters and Mousehole column as well as remove a number of articles. Just look to the last page of this book for the list of articles *not* in this book.



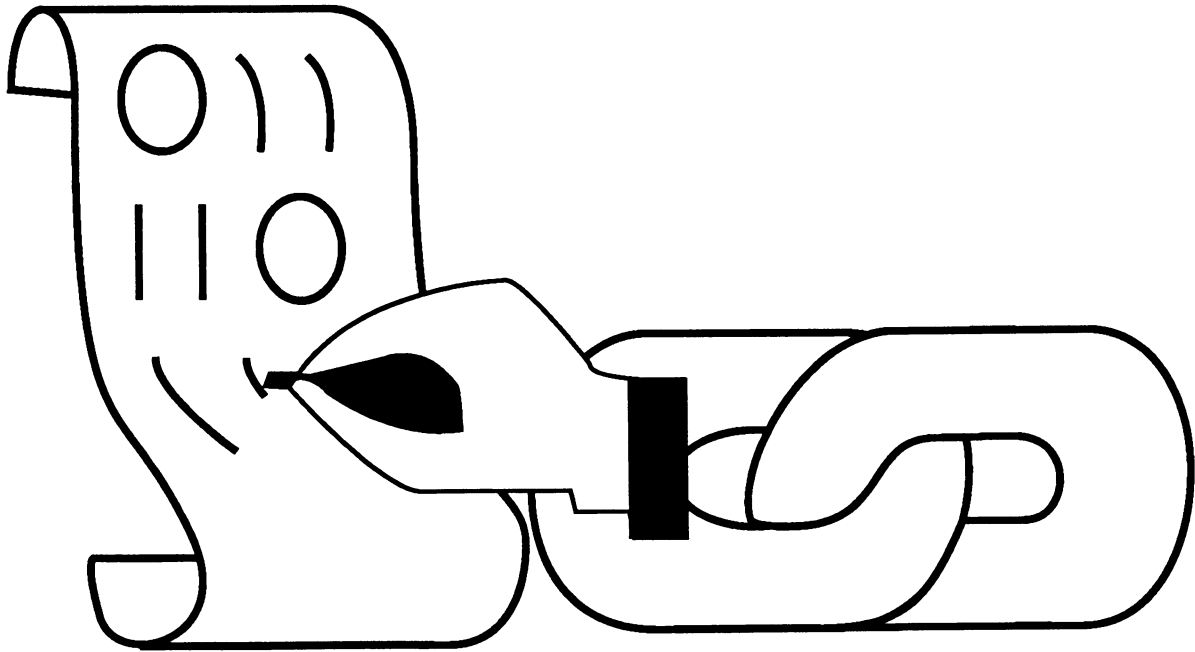
Table of Contents

The Best of MacTutor, Volume 5, 1989

Title	(Source Disk)	Page	Title	(Source Disk)	Page
Table of Contents		v...vi	List of Controls LDEF	(45)	371
Assembly		1...47	Build a Network Painting Program	(46)	379
Help! An Easy to use Help Function	(40)	2	A Window Menu	(46)	400
Convert PICTs to Regions	(45)	11	Introduction to MacApp	(47)	409
MDS->MPW		17	New CDEF Messages		418
Cerial (Serial) I/O	(46)	19	A Look at the PREC Resource	(47)	420
Real-Time Driver	(49)	31	A Tale of Two Quadratic Plotters, Part I	(47)	432
Assembly Language for the Rest of Us	(51)	38	A Tale of Two Quadratic Plotters, Part II	(48)	454
			'snd ' Zounz Great!	(48)	475
C		48...244	Understanding Color Icons	(49)	492
Bezier Curve Ahead!	(40)	49	Writing INITs in Pascal	(49)	501
Event Simulator	(40)	54	Dispatch Aliasing	(50)	503
How to Write a Spreadsheet in LSC	(42)	58	AACK Gives AppleTalk Shell	(51)	514
All About Scrolling Windows	(43)	69	Capture Color Icons	(51)	528
April Fool's INIT: Scrolling the Menu Bar!?	(43)	88			
Segmenting DA's	(44)	93	Forth		532...557
Change the Boot Disk Icon	(44)	98	System Queues	(40)	533
Lisa's Hourglass Is Back!	(44)	100	Record Definitions	(41)	539
Window Structures	(45)	107	System CDEV	(44)	545
Sub and Superscripting with TE	(45)	110	INITs and Patches	(45)	550
Extend Your Favorite Editor	(46)	122	Appletalk Protocol Handlers	(46)	554
Pseudo Objects	(47)	129			
C++ Overview		138	Basic		558...589
Three Doodats	(48)	142	Time Retrieval And Storage Techniques	(40)	559
World CDEV Investigated	(48)	146	ADB Demo in ZBasic 5.0 & LS Pascal 2.0	(42)	562
Dot Matrix Printer Driver	(49)	149	Pop Up Menus in True Basic	(43)	575
Writing INITs in THINK C	(49)	174	Sounding Off with MS QuickBasic	(44)	578
Apple's MIDI Manager	(50)	181	Help for HELP Software	(45)	583
Inside Controls	(50)	190	Useful Routines For ZBasic	(46)	587
Alternate Scroll Bar CDEF	(50)	213			
Apple's C++	(50)	222	HyperChat		590...685
AppleTalk Chatter Box	(51)	227	HyperAppleTalk	(40)	591
C++ Sample Application	(51)	239	A Real Window XCMD	(41)	599
			XCMD Corner: HyperAppleTalk, Part II	(41)	611
Pascal		245...531	AppleTalk DSP Part III	(42)	619
Finder Icon Controls	(40)	246	XCMD Corner: Window Scrollbars	(42)	626
Security Patrol for Viruses	(41)	261	XCMD Corner: HyperAppleTalk IV	(43)	643
How To Write a Font Tool for MPW	(42)	289	Menu Bar XCMD	(44)	651
Re-sizing with Window Corners	(43)	295	XCMDs in THINK C	(44)	653
Designer CDEFs	(43)	309	XCMD Library	(45)	659
MPW Tools with TML Pascal II	(44)	319	HyperArrays	(45)	662
The Notification Manager	(44)	330	Sector Dumps	(46)	667
MIDI User Interface	(44)	336	HFS Tree Climbing	(47)	671
Modeless Search and Replace	(44)	344	The Blessed Folder	(48)	674
Have Your Window Do The Splits	(45)	354	CopyFile	(49)	677
Spray Can and Paint Bucket Tools	(45)	366	XCMD Corner: Catalog XCMD	(50)	679
			A Look Into Networking	(51)	681

Title	(Source Disk)Page	Title	(Source Disk)Page
More Articles	686...715	Mousehole Report	733...754
20 Steps to Printing Incompatibility	687	Topical Index	756...759
Developer Conference Report	693	Author's Index	760...761
The Installer Explored	(47) 697	Articles not included in this book	762
Compiler Comparisons	(47) 702		
SADE Debugging	(48) 707		
New Typographic Frontier	712		
Letters	716...732		

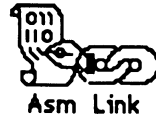




Assembly Lab

Assembly Lab

Help! An Easy to use Help function



John Holder
Blue Lake, CA
Volume 5, Number 1

What's this all about?

This article will show you a way of presenting a Help function (hereafter called DoHelp) for any of your programs that is very easy to implement and use. DoHelp must be XREF'ed (XREF DoHelp) at the beginning of the source file that calls it. To use it I just add a 'Help...' menu in an appropriate place in your application (I put it right below the standard 'About Application...' menu item in the apple menu). Call it by simply doing a BSR DoHelp when the item has been selected by the user.

DoHelp was created with the Macintosh 68000 Development System. First assemble the file 'DoHelp.Asm', then run the linker on your applications Linker file to Link DoHelp with the rest of your code, and finally run RMaker on your Rmaker file. You can create the needed 'help' resources with ResEdit (or whatever you prefer) and add them to your applications resource file. 'YourApp.Link' & 'YourApp.R' are shown as examples of how you link your applications code with DoHelp's code.

The way DoHelp works is to show a list of resource names (a Help topic list created by showing the names of each 'help' resource in numeric order) using the List Manager. This makes adding and editing Help topics very easy, by using ResEdit or compiling the needed resources with RMaker. The user can click on any name to access its contents, which is displayed in a scrollable text window (which temporarily replaces the list of Help topics). When the user is done reading this, he can click on the button or within the text itself to return to the Help topics. From here he can choose to look at another topic or select the 'Done with Help' button to return to the application.

Why?

I wrote DoHelp to put into my latest shareware application called 'Form It!'. I wanted to keep it as generic as possible so I could use it for any other applications that I might create in the future. I made it so the calling application wouldn't need to know anything specific about the help but would just have to call it when the user needs it. It handles everything else.

What's happening , step by step.

First, all files needed are Included at the beginning and a couple of often used routines for saving and restoring the registers are defined as Macros. A few standard EQUates are also defined along with some values used by the List Manager. The last EQUate is the resource ID# of the DLOG used to display the Help function. Finally the DoHelp routine is XDEF'ed so the Linker will know what the heck is going on when it's being linked to the code that calls it!

Now a dialog box is created, the default button is highlighted,

and we jump to the routine to set up the list of named 'help' resources. To accomplish this, we first calculate the useable width of the cells used in the list by checking the size of the User Item's rectangle ($\text{right-left-scrollbarwidth} = \text{widthofcells}$) of the Help dialog box (item #2 in this DLOG). Then we create a new list with _LNew and set its selflags field to allow only one item to be selected at a time.

The routine to add each named 'help' resource is then called. We count how many resources of this type there are with _CountResources and set list drawing off so it won't show each item as it's being added to the list. Starting from the last resource we cycle through all the resources and add their names to the list with _LAddRow & _LSetCell. After the last resource (the resource id#1 since we're adding in reverse) has been put in the list, we exit the loop and turn list drawing on and update the list to draw it.

The Dialog filter

After everything is initialized we go into a loop to watch for dialog events. A Dialog Filter is used to handle the List Manager events, all other dialog events are handled by _ModalDialog. The filter is set up to get values off of the stack (passed by _ModalDialog) and create some local variables by using the LINK instruction to set up a stack frame. When a click occurs in the rectangle of the User Item, _LClick is called to handle scrolling or a selection. If a help topic (list item) is clicked on, the rectangle is erased, we change the buttons (item #1) name to an appropriate message ('Done Viewing'), the cursor is changed to the watch cursor and we jump to the Do_Help_View routine.

A new Text Edit record is made with _TENew and its text size and font is set. The rectangle around the User Item is framed and a scroll bar is created. The selected cells data is retrieved using _LGetCell to get the name of the resource to read in with _GetNamedResource. If the given resource doesn't exist (why it wouldn't I don't know, but it pays to be safe!) we dispose of the TE record and the scroll bar and return to wait for the next event. Otherwise, we copy the data in the 'help' resource into the Text Edit Record and calculate how many lines are in the data. If there are more lines than one screenfull, the scroll bar is set up to match the amount of lines in the data. Next, we jump to the routine to handle text scrolling using another Dialog Filter to handle it.

If the scroll bar is clicked on, we check which part of the scroll has been clicked and handle each part individually. If the Thumb is clicked, it's tracked by _TrackControl until released and then we scroll the text according to the difference between the controls old and new values. The text is scrolled a line at a time or a pagefull at a time depending on the part clicked on. The routine Scroll_The_Text figures how many pixels to scroll by multiplying the height of a line of text (found in the teLineHite

field of the TE record) by how many lines of text are to be scrolled.

When either the button or the content of the text in the dialog is clicked, the TE record and the scroll bar are disposed of, the list is updated to show it again, we change the buttons name back to what it was and we return to watch for more list events. If the user clicks on another list item, the whole process starts again. Otherwise if the user clicks the button or hits Return, the dialog and the list handle are disposed of and control is returned to the calling application.

Credits

The basis of some of the routines used in DoHelp (such as the dialog filters and scroll handling routines) are from Dan Weston's extremely useful books "The Complete Book of Macintosh Assembly Language Programming" (volumes I and II). These books have proved very helpful to my learning to use the power of the Mac with Assembly language!

The End...

There are many ways that DoHelp could be enhanced. For instance, a search feature or a way to print selected help topics could be added. The source code is well commented, so anything that this article doesn't explain, the comments should be able clear up. Well, that's it, I hope you find this useful enough to use in your own applications!

```

;-----
;FileName: DoHelp.Asm
;(C) 1988 by John Holder
;-----
;What it does:
;A generic Help routine:
;Allows user to click in a list of names (these names are
;actually the names of 'help' resources), and show the data
;contained in this resource (ASCII text), in a scrollable
;window. Nothing about the dialog (other than its id#) is
;known, so the dialog can be modified without changing this
;code. To make topics for your help routine, just create
;NAMED resources of type 'help' (lowercase) in your
;application starting at id#1 and making each id# one more
;than the last one. To create the appearance of subtopics
;just add a few spaces to the beginning of the names of
;the appropriate 'help' resources.

```

```

;To use this in your application, just make sure you:
;XREF DoHelp
;at the beginning of your source code and you call it by
;doing a:
;    BSR DoHelp
;in the appropriate place in your code
Include Traps.D      ; Traps
Include ToolEqu.D    ; ToolBox equates
Include SysEqu.D     ; System equates
Include FSEqu.D      ; File equates
Include PackMacs.Txt ; PACKage mgr equates
Include QuickEqu.D   ; QDraw equates

```

```

MACRO SaveRegs =
    MOVEM.L    A0-A4/D0-D7,-(SP)
|
MACRO RestoreRegs =
    MOVEM.L    (SP)+,A0-A4/D0-D7
|
true          equ $0100
false         equ 0

```

```

nil          equ 0
arraycolumns equ 1
arrayrows    equ 0
celldepth    equ 16 ;depth of Cells in List
modify       EQU 14 ;State of keys and button
message      equ 2  ;Message returned in EventRecord
DialogID     equ 2000 ;res id# of DLOG resource

```

XDEF DoHelp ;Define function DoHelp for Linker

```

;-----
; <<<< The beginning of the DoHelp routine >>>>
;-----

```

```

DoHelp
    saveregs
    CLR.L    -(SP) ;space for ptr
    move.w   #DialogID,-(sp)
    CLR.L    -(SP) ;wstorage
    MOVE.L    #-1,-(SP)
    _GetNewDialog
    MOVE.L    (SP),DialogPtr(A5) ;save ptr
    _SetPort
    ;set up stack for my routine
    move.l    DialogPtr(A5),-(sp)
    move      #1,-(sp)
    bsr      HighlightDialogButton
    ;Set up list of all 'help' resources
    bsr      SetUpHelpList
    _InitCursor

```

```

Dialog_Loop
    pea      DialogsFilter ;use filter to watch
    pea      ItemNumber(A5) ;for List mgr events
    _ModalDialog

```

```

WhatsTheHaps21
    ;which item was clicked
    cmp.w    #1,ItemNumber(A5)
    beq      Quit_Help ;click in 'Done..' but
    bra      Dialog_Loop ;loop until done

```

```

Quit_Help
    ;all done, lets get outta here!
    ;dispose of the List handle
    move.l    ListHandle(A5),-(sp)
    _Dispose
    ;kill the dialog
    move.l    DialogPtr(A5),-(sp)
    _DisposDialog
    ;return to calling application!
    restorerregs
    rts

```

```

;-----
;<< Dialog Filter to watch for list selections in User Item >>
;-----

```

```

DialogsFilter
    ;this routine expects parameters to be on the stack (handled
    ;by _ModalDialog)
;PROCEDURE
DialogsFilter (thedialog:dialogptr;VAR theEvent:EventRecord
;            VAR itemhit:integer): boolean

```

```

Parambytes    SET 12
tItemHit      SET 8 ;a ptr to an int!
tEvent        SET 12 ;event rec
tDialog       SET 16 ;dialog ptr
result        SET 20 ;result returned
iPoint        SET -4 ;mouse point
itype         SET -6 ;item type
iHdl          SET -10 ;item hand
iBox          SET -18 ;item rect
localbytes    SET -18

```

```

link    A6,#localbytes
saveregs
;get User Item info (rect)
move.l  tDialog(A6),-(sp)
move    #2,-(sp)
pea     iType(A6)
pea     iHdl(A6)
pea     iBox(A6)
_GetDItem
move.l  tevent(A6),A0
move    evtnum(A0),D0
cmp     #KeyDwnEvt,D0 ;was it a key down?
beq     CheckForEnterorReturn ;yep!
move.l  tevent(A6),A0
move    evtnum(A0),D0
cmp     #mButDwnEvt,D0 ;was it a mouse down?
bne     LetDialogHandleIt ;if not, let
                                ;_ModalDialog
                                ;handle
;was it a mouse click in the List Box (User Item)?
lea     evtMouse(A0),A0
lea     iPoint(A6),A1
move.l  (A0)+(A1)+
pea     iPoint(A6)
_GlobalToLocal
clr     -(sp)
move.l  iPoint(A6),-(sp)
pea     iBox(A6)
_PtInRect
move    (sp)+,D0
beq     LetDialogHandleIt ;not in user item
bsr     HandleListEvent ;handle list event!
;we've taken care of the event
move.l  tItemHit(A6),A0
move    #nil,(A0) ;set itemhit to nil
move    #true,result(A6) ;stop _ModalDialog
bra     FilterExit ;from handling!

LetDialogHandleIt
move    #nil,result(A6)

FilterExit
;restore stack to the way it was before
;and return
restoreregs
unlk    A6
move.l  (sp)+,A0
adda    #parambytes,SP
jmp     (A0) ;rts

;
; <<<< Check for Return or Enter key press >>>>
;
CheckForEnterorReturn
;if the Enter or Return key was pushed, set the itemhit to #1
;and set result to false so it will be handled by _ModalDialog

move.l  tevent(A6),A0
move.l  message(A0),D0
cmp.b   #$0d,D0 ;return key?
beq     SetResult ;yep!
cmp.b   #$03,D0 ;Enter key?
beq     SetResult ;yep!
bra     LetDialogHandleIt ;neither key

SetResult
move.l  tItemHit(A6),A0
move    #1,(A0) ;set itemhit to 1
move    #true,result(A6) ;let _ModalDialog
                                ;handle it
bra     FilterExit

;
; <<<< Click in List box! >>>>
;

```

```

;handle a click in the list box area
HandleListEvent

```

```

;_Click will handle scrolling & selection of list
;items
pea     iPoint(A6)
_GetMouse
clr     -(sp)
move.l  iPoint(A6),-(sp)
move    tEvent+Modify(A6),-(sp)
move.l  listhandle(A5),-(sp)
_Click
move    (sp)+,D0
clr.l   TheCell(A5) ;start at cell 0,0
                                ;for _GetSelect
clr     -(sp)
move    #true,-(sp)
pea     TheCell(A5)
move.l  listhandle(A5),-(sp)
_GetSelect
move    (sp)+,D0
beq     Click_Not_In_Cell ;no cell selected
;unselect the selected cell
move    #false,-(sp)
move.l  TheCell(A5),-(sp)
move.l  listhandle(A5),-(sp)
_SetSelect
;de-activate the list
move    #false,-(sp)
move.l  listhandle(A5),-(sp)
_Activate
bsr     Erase_Rect ;clear topic list
bsr     Set_To_DoneViewing ;change button name
bsr     GetandSetWatch ;show watch cursor
bsr     Do_Help_View ;show text from
                                ;'help' res
;re-activate the list
move    #true,-(sp)
move.l  listhandle(A5),-(sp)
_Activate

```

```

Click_Not_In_Cell
rts

```

```

;
; <<<< Show the user selected help text >>>>
;
;open the named resource and view its contents
Do_Help_View

```

```

;Use all of rect except enough room for scroll bar
;for text window
bsr     Get_Rect_Of_Item
lea     DispRect(A5),A2
sub     #16,Right(A2)
;create new Text edit record
clr.l   -(sp)
pea     DispRect(A5)
pea     DispRect(A5)
_TENew
move.l  (sp)+,TextHand(A5) ;store handle
;set TE rec. text to 9 point Monaco
move.l  TextHand(A5),A1
move.l  (A1),A1
move    #9,teSize(A1) ;Text Size!
move.l  TextHand(A5),A1
move.l  (A1),A1
move    #Monaco,teFont(A1) ;Text Font!
bsr     Frame_The_Rect

bsr     Get_Rect_Of_Item
;figure rectangle for scroll bar!
lea     DispRect(A5),A2
move    Right(A2),D2

```

```

sub    #15,D2      ;scroll bar width
move   D2,Left(A2) ;left = right - 15
add    #1,Right(A2);right = right + 1
                        ;we do because the
                        ;text rect is 1
                        ;pixel inside the
                        ;User Item's rect
pea     DispRect(A5);expand top & bottom
move   #0,-(sp)    ;of rect by 1 pixel
move   #-1,-(sp)
_InsetRect
;create scroll bar!
clr.l   -(sp)
move.l  DialogPtr(A5),-(sp)
pea     DispRect(A5) ;scrolls rect
move.l  #nil,-(sp)
move   #true,-(sp) ;visible
move   #0,-(sp) ;init value
move   #0,-(sp) ;min value
move   #0,-(sp) ;max value(start at 0)
move   #16,-(sp) ;proc id (16=scroll)
clr.l   -(sp) ;refCon
_NewControl
move.l  (sp)+,ScrollHand(A5) ;save handle
move   #255,DataLength(A5) ;max length of
                        ;data allowed

;Get the selected cells data
lea     AString(A5),A0 ;pt to byte after
add.l   #1,A0 ;length byte
move.l  A0,-(sp) ;put ptr on stack
pea     DataLength(A5) ;data length var
move.l  TheCell(A5),-(sp)
move.l  ListHandle(A5),-(sp)
_GetCell
;add length of data to String ptr
lea     AString(A5),A0
move   DataLength(A5),D0
move.b  D0,(A0)
;get the res. & load it
clr.l   -(sp)
move.l  #'help',-(sp)
pea     AString(A5)
_GetNamedResource
move.l  (sp),ResHand(A5)
_LoadResource
;just to make sure the resource exists!
move.l  ResHand(A5),A0
cmp.l   #nil,A0 ;does res. exist?
bne     A_0k ;yep, it's there, go on
bsr     DumpStuff ;no such resource!
rts

A_0k
move.l  ResHand(A5),A0 ;lock handle
_HLock
;size of handle (text data)
move.l  ResHand(A5),A0
_GetHandleSize
move.l  D0,D2 ;D2 = length of text data
;set up Text Edit Record to use 'help' data
move.l  ResHand(A5),A0 ;need text ptr
move.l  (A0),-(sp) ;convert handle
move.l  D2,-(sp) ;text length
move.l  TextHand(A5),-(sp)
_TESetText
move.l  ResHand(A5),A0 ;unlock handle
_HUnlock
;see how many lines of text are in the
;'help' data
move.l  TextHand(A5),A1
move.l  (A1),A1
move   tenLines(A1),d3 ;D3= lines of text!
;see how many lines fit into rect of user
;item of dialog box

```

```

bsr     CalcLines_In_One_Windowful
move   LinesInWind(A5),D1
sub     D1,D3 ;sub enough for a
cmp     #nil,D3 ;window full!
ble     Wont_Pass_Bottom ;not enough lines
                        ;to add a scroll bar
;set the scroll bars max value by the # of lines
move.l  ScrollHand(A5),-(sp)
move   D3,-(sp) ;# of lines
_SetMaxCtl

```

```

Wont_Pass_Bottom
;update the text
pea     DispRect(A5)
move.l  TextHand(A5),-(sp)
_TEUpdate
_InitCursor
;go watch for text events
bsr     HandleTextEditDialogEvent
rts     ;done with the text, return

; <<<< Handle events in Dialog box (with text scrolling)! >>>>
HandleTextEditDialogEvent
saveregs
D_Loop
pea     CheckTextScroll ;filter to watch
pea     ItemNumber(A5) ;for text scrolling
_ModalDialog

;check which buttons are clicked here
cmp.w   #1,ItemNumber(A5)
beq     Done_With_This ;done viewing text

;if ItemNumber(A5) = 2, that means the user
;clicked in the content of the text of the dialog
;which is handled the same as clicked the
;'Done Viewing' button
cmp.w   #2,ItemNumber(A5)
beq     Done_With_This ;done viewing text
bra     D_Loop ;keep looping

; <<<< Finished here with text view >>>>
Done_With_This
bsr     DumpStuff
restoregs
rts

; <<<< Dump TE rec. & scroll bar >>>>
DumpStuff
;dispose of text edit record
move.l  TextHand(A5),-(sp)
_TEDispose
;dispose of Scroll bar
move.l  ScrollHand(A5),-(sp)
_DisposControl
bsr     Erase_Rect ;erase the rect
;reset text font & size to System values
move   #12,-(sp) ;12 point
_TextSize
move   #0,-(sp) ;Use System Font
_TextFont
;re-show the list by causing list update
MOVE.L  DialogPtr(A5),A1
move.l  24(A1),-(sp)
move.l  ListHandle(A5),-(sp)
_Update
bsr     Frame_The_Rect ;re-frame list rect
bsr     Set_To_DoneWithHelp
rts

; <<<< Dialog Filter to watch for text scrolling >>>>

```

```

;
CheckTextScroll
;this routine expects parameters to be on the stack (set up
;by _ModalDialog)
;PROCEDURE
;CheckTextScroll (thedialog:dialogptr;VAR theEvent:EventRecord
;VAR itemhit:integer): boolean

```

```

Parambytes SET 12
tItemHit SET 8 ;a ptr to an int!
tEvent SET 12 ;event rec
tDialog SET 16 ;dialog ptr
result SET 20 ;result returned

```

```

iPoint SET -4 ;mouse point
itype SET -6 ;item type
iHdl SET -10 ;item hand
iBox SET -18 ;item rect
PartCode SET -20 ;used for scroll
localbytes SET -20

```

```

link A6,#localbytes
saveregs
move.l tDialog(A6),-(sp)
move #2,-(sp)
pea itype(A6)
pea iHdl(A6)
pea iBox(A6)
_GetDItem
;only rect in user item will be scroll bar that
;this dialog filter handles
lea iBox(A6),A2
move Right(A2),D2
sub #16,D2 ;scroll bar width
move D2,Left(A2) ;left = right - 16
move.l tevent(A6),A0
move evtnum(A0),D0
cmp #KeyDownEvt,D0 ;was it a key down?
beq CheckForEnterorReturn2;yes!
move.l tevent(A6),A0
move evtnum(A0),D0
cmp #mButDwnEvt,D0 ;was it a mouse down?
bne LetDialogHandleIt2 ;if not, let
;_ModalDialog
;handle

```

```

;change mouse pt into local coordinates
lea evtMouse(A0),A0
lea iPoint(A6),A1
move.l (A0)+(A1)+
pea iPoint(A6)
_GlobalToLocal
;was it a mouse click in the Text Box scroll
;bar (User Item)?
clr -(sp)
move.l iPoint(A6),-(sp)
pea iBox(A6)
_PtInRect
move (sp)+,D0
beq LetDialogHandleIt2 ;not in scroll,
;let dialog handle

```

```

bsr HandleScroll ;go handle text
;scroll event!
;we've taken care of the event
move.l tItemHit(A6),A0
move #nil,(A0) ;set itemhit to nil

```

```

move #true,result(A6) ;set result to true,
bra FilterExit2 ;stop _ModalDialog
;from handling

```

```

LetDialogHandleIt2
move #nil,result(A6) ;set to nil, let
;_ModalDialog handle
;it

```

```

FilterExit2
;get out of the routine here!
restoreregs
unlk A6
move.l (sp)+,A0
adda #parambytes,SP
jmp (A0) ;rts

```

```

; <<<< Check for Return or Enter key press >>>>

```

```

CheckForEnterorReturn2

```

```

;if the Enter or Return key was pushed set the itemhit to #1
;and set result to false so it will be handled!

```

```

move.l tevent(A6),A0
move.l message(A0),D0
cmp.b #$0d,D0 ;return key?
beq SetResult2 ;yes!
cmp.b #$03,D0 ;Enter key?
beq SetResult2 ;yes!
bra LetDialogHandleIt2 ;neither key

```

```

SetResult2

```

```

move.l tItemHit(A6),A0
move #1,(A0) ;set itemhit to 1
move #true,result(A6) ;let modal dialog
bra FilterExit2 ;handle it

```

```

; <<<< ALL text scrolling routines >>>>

```

```

; <<<< Click in Text Scroll bar >>>>

```

```

HandleScroll

```

```

pea iPoint(A6)
_GetMouse
clr -(sp)
move.l iPoint(A6),-(sp)
move.l DialogPtr(A5),-(sp)
pea WhichControl(A5)
_FindControl
move (sp)+,PartCode(A6)
beq ScrollDone ;not in any control
;was click in the scroll bar?
;compare value returned by _FindControl & the
;scrolls handle we have stored, if they are
;equal then do the scroll routines
move.l ScrollHand(A5),D0
move.l WhichControl(A5),D1
cmp.l D0,D1
beq Click_In_Scroll
bra ScrollDone ;click not in scroll

```

```

Click_In_Scroll

```

```

;which part of scroll?
cmp #inUpButton,PartCode(A6)
beq DoUpScroll
cmp #inDownButton,PartCode(A6)
beq DoDownScroll
cmp #inPageUp,PartCode(A6)
beq DoPageUpScroll
cmp #inPageDown,PartCode(A6)
beq DoPageDownScroll
cmp #inThumb,PartCode(A6)
beq DoThumb
bra ScrollDone

```

```

;handle different parts of scroll
DoUpScroll

```

```

clr -(sp)
move.l WhichControl(A5),-(sp)

```



```

move.l iPoint(A6),-(sp)
pea UpActionProc
_TrackControl
move (sp)+,D0
bra ScrollDone

DoDownScroll
clr -(sp)
move.l WhichControl(A5),-(sp)
move.l iPoint(A6),-(sp)
pea DownActionProc
_TrackControl
move (sp)+,D0
bra ScrollDone

DoPageUpScroll
clr -(sp)
move.l WhichControl(A5),-(sp)
move.l iPoint(A6),-(sp)
pea PageUpActionProc
_TrackControl
move (sp)+,D0
bra ScrollDone

DoPageDownScroll
clr -(sp)
move.l WhichControl(A5),-(sp)
move.l iPoint(A6),-(sp)
pea PageDownActionProc
_TrackControl
move (sp)+,D0
bra ScrollDone

;handle click in thumb of control
DoThumb
clr -(sp)
move.l WhichControl(A5),-(sp)
_GetCtlValue
move (sp)+,D4 ;D4 = old control value
clr -(sp)
move.l WhichControl(A5),-(sp)
move.l iPoint(A6),-(sp)
clr.l -(sp)
_TrackControl
move (sp)+,D0
clr -(sp)
move.l WhichControl(A5),-(sp)
_GetCtlValue
move (sp)+,D3 ;D3 = new cntl value
sub D3,D4 ;D4-D3=new view value
move D4,D6
bsr Scroll_The_Text ;scroll text selected spot
bra ScrollDone

;return from HandleScroll
ScrollDone
rts

;
; <<<< Handle Up Scroll arrow >>>>
;

UpActionProc
;offsets into A6 stack
thecontrol set 10
pcode set 8
parambytes set 6

link A6,#0
saveregs
tst pcode(A6)
beq UpActionDone
clr -(sp)
move.l WhichControl(A5),-(sp)
_GetCtlValue
move (sp)+,D1
beq UpActionDone ;do nothing if control

```

```

;value is already 0
sub #1,D1 ;sub 1 from scroll val
move.l WhichControl(A5),-(sp)
move D1, -(sp)
_SetCtlValue ;set to new value
move #1,D6 ;d6 = how many lines
bsr Scroll_The_Text ;to scroll!

```

```

UpActionDone
restoreregs
unlk A6
move.l (sp)+,A1
adda.w #parambytes,SP
jmp (a1) ;rts

;
; <<<< Handle Down Scroll arrow >>>>
;

```

```

DownActionProc
;offsets into A6 stack
thecontrol set 10
pcode set 8
parambytes set 6

link A6,#0
saveregs
tst pcode(A6)
beq DownActionDone
clr -(sp)
move.l WhichControl(A5),-(sp)
_GetCtlValue
move (sp)+,D2
add #1,D2 ;add 1 to scroll val
clr -(sp)
move.l WhichControl(A5),-(sp)
_GetMaxCtl
move (sp)+,D1
;don't scroll past controls max setting!
cmp D1,D2
bgt DownActionDone
move.l WhichControl(A5),-(sp)
move D2, -(sp)
_SetCtlValue
move #-1,D6 ;d6 = how many lines
bsr Scroll_The_Text ;to scroll!

```

```

DownActionDone
restoreregs
unlk A6
move.l (sp)+,A1
adda.w #parambytes,SP
jmp (a1) ;rts

;
; <<<< Handle PageUp Scroll arrow >>>>
;

```

```

PageUpActionProc
;offsets into A6 stack
thecontrol set 10
pcode set 8
parambytes set 6

link A6,#0
saveregs
tst pcode(A6)
beq PageUpActionDone
clr -(sp)
move.l WhichControl(A5),-(sp)
_GetCtlValue
move (sp)+,D1
move LinesInWind(A5),D0
sub #1,D0
sub D0,D1
move D0,D6 ;d6 = how many lines
;to scroll!
;are we going to go before 1st line?

```

```

; if so, figure how many lines from current position
; until 0 (zero) and use that # to scroll text!
cmp     #nil,D1
bge     Not_Past_Begin2
add     D1,D6           ;set to proper value
                        ;for scroll
move    #nil,D1        ;to set ctrl

Not_Past_Begin2
move.l   WhichControl(A5),-(sp)
move     D1,-(sp)
_SetCtlValue
bsr      Scroll_The_Text

PageUpActionDone
restoreregs
unlk     A6
move.l   (sp)+,A1
adda.w   #parambytes,SP
jmp      (a1) ;rts

;
; <<<< Handle PageDown Scroll arrow >>>>
;
PageDownActionProc
; offsets into A6 stack
thecontrol set 10
pcode set 8
parambytes set 6

link     A6,#0
saveregs
tst      pcode(A6)
beq      PageDownActionDone
clr      -(sp)
move.l   WhichControl(A5),-(sp)
_GetCtlValue
move     (sp)+,D7
move     D7,D2
move     LinesInWind(A5),D0
sub      #1,D0
add      D0,D2
neg      D0
move     D0,D6         ;d6 = how many lines
                        ;to scroll!
; are we going to go after last line?
; if so, figure how many lines from current position
; until last line and use that # to scroll text
clr      -(sp)
move.l   WhichControl(A5),-(sp)
_GetMaxCtl
move     (sp)+,D3 ;to set ctrl
cmp      D3,D2
ble      Not_Past_End
sub      D3,D7         ;set to proper value
move     D7,D6         ;for scroll
move     D3,D2         ;to set ctrl

Not_Past_End
move.l   WhichControl(A5),-(sp)
move     D2,-(sp)
_SetCtlValue
bsr      Scroll_The_Text

PageDownActionDone
restoreregs
unlk     A6
move.l   (sp)+,A1
adda.w   #parambytes,SP
jmp      (a1) ;rts

;
; <<<< Do the actual text scrolling >>>>
;
Scroll_The_Text
; d6 has how many lines and which direction to scroll

```

```

; (depending on whether the value is pos. or neg.)
; use teLineHite to figure how many pixels to scroll
; by multiplying it with the amount of lines
clr.l    D2
move.l    TextHand(A5),A2
move.l    (A2),A2
move      teLineHite(A2),D2
muls      D6,D2
move      #nil,-(sp) ;no horiz. scroll
move      D2,-(sp) ;vert. scroll
move.l    TextHand(A5),-(sp)
_TESCroll
rts

; <<<< END of all text scrolling routines >>>>
;
; <<<< How many text lines can fit in user item rect >>>>
;
Calc_Lines_In_One_Windowful
saveregs

bsr      Get_Rect_Of_Item
clr.l    D4
lea      DispRect(A5),A2
move     Top(A2),D3
move     Bottom(A2),D4
sub      D3,D4         ;Top-Bottom = how many
                        ;pixels in text view box
                        ;D4= height (in pixels)
                        ;of help window

clr.l    D5
move.l    TextHand(A5),A2
move.l    (A2),A2
move      teLineHite(A2),D5 ;rect height
divu      D5,D4         ;divided by the
                        ;height of one
                        ;line of text =
                        ;how many lines
                        ;can fit!
; D4 now = how many lines of text can fit into one
; screen of the user item of the dialog box!
move     D4,LinesInWind(A5)
restoreregs
rts

;
; <<<< Erase inside of user item rect of dialog >>>>
;
Erase_Rect
; erase entire user item rectangle
bsr      Get_Rect_Of_Item
pea      DispRect(A5)
_EraseRect

rts

;
; <<<< Get user item rect of dialog >>>>
;
Get_Rect_Of_Item
; Get info from User Item #2
move.l    DialogPtr(A5),-(SP)
move      #2,-(sp)
pea      ItemType(A5)
pea      ItemHandle(A5)
pea      DispRect(A5)
_GetDItem
rts

;
; <<<< Change button to say 'Done Viewing' >>>>
;
Set_To_DoneViewing
bsr      Get_Button_Handle
move.l    ItemHandle(A5),-(sp)
pea      DoneViewing
_SetCtlTitle
rts

```

```

;
; <<<< Change button to say 'Done with Help' >>>>
;
Set_To_DoneWithHelp
    bsr    Get_Button_Handle
    move.l  ItemHandle(A5),-(sp)
    pea    DoneString
    _SetTitle
    rts

;
; <<<< Get handle of button in dialog >>>>
;
Get_Button_Handle
    ;Get the buttons handle (for changing the buttons
    ;title)
    move.l  DialogPtr(A5),-(sp)
    move    #1,-(sp)
    pea    ItemType(A5)
    pea    ItemHandle(A5)
    pea    DispRect(A5)
    _GetDItem
    rts

;
; <<<< Get watch CURSor and display it >>>>
;
GetandSetWatch
    clr.l   -(sp)
    move    #watchcursor,-(sp)
    _GetCursor
    move.l  (sp)+,A0
    move.l  (A0),-(sp)
    _SetCursor
    rts

;
; <<<< Frame the user item rect of dialog >>>>
;
Frame_The_Rect
    ;frame the outside of the user items rectangle
    bsr    Get_Rect_Of_Item
    pea    DispRect(A5)    ;Inset rect by 1
    move    #-1,-(sp)      ;outward
    move    #-1,-(sp)
    _InSetRect
    pea    DispRect(A5)
    _FrameRect
    rts

;
; <<<< Hilight A Dialog Button >>>>
;
;this routine expects parameters to be on the stack
;PROCEDURE HilightDialogButton (WPtr: Ptr; WhichItem: Int)
;Will hilight a button in a dialog box
HilightDialogButton
    WindPt    SET 10
    WhichItem SET 8
    parambytes SET 6

    ;local variables
    TheType    SET -4 ;Vars for _GetDItem
    TheHandle   SET -8 ;handle
    TheRect     SET -16 ;item rect
    localbytes SET -16 ;for link

    link    A6,#localbytes
    saveregs
    move.l  WindPt(A6),-(sp)
    _SetPort
    move.l  WindPt(A6),-(sp)
    move    WhichItem(A6),-(sp)
    pea    TheType(A6)
    pea    TheHandle(A6)
    pea    TheRect(A6)
    _GetDItem
    move    #3,-(sp)

```

```

    move    #3,-(sp)
    _PenSize
    pea    TheRect(A6)
    move    #-4,-(sp)
    move    #-4,-(sp)
    _InSetRect
    pea    TheRect(A6)
    move    #16,-(sp)
    move    #16,-(sp)
    _FrameRoundRect
    move    #1,-(sp)
    move    #1,-(sp)
    _PenSize
    restoreregs
    unlk    A6
    move.l  (sp)+,A0
    adda    #parambytes,SP
    jmp     (A0) ;rts

;
; <<<< Set up the dialog box with the help info >>>>
;
SetUpHelpList
    saveregs
    bsr    Calc_Cell_Width
    bsr    Get_Rect_Of_Item
    ;must allow room for the lists scroll bar on the right
    ;side so subtract 15 from the rects right side!
    lea    DispRect(A5),A1
    sub    #15,right(A1)
    ;set up the arrayrect (for the cells rows and columns)
    ;for the List Mgr
    lea    arrayrect(A5),a0
    move.l  #0,(A0)+
    move    #arrayRows,(A0)+
    move    #arraycolumns,(A0)+
    move    #celldepth,D0
    swap    D0
    move    cellWidth(A5),D0
    ;create a new List
    clr.l   -(sp)
    pea    DispRect(A5)    ;list rect!
    pea    arrayrect(A5)
    move.l  D0,-(sp)      ;cell size
    move    #0,-(sp)      ;res id of proc.
    move.l  DialogPtr(A5),-(sp) ;window ptr
    move.w  #false,-(sp)   ;draw it?
    move    #false,-(sp)   ;has grow?
    move    #false,-(sp)   ;horiz scroll?
    move    #true,-(sp)    ;vert scroll?
    _New
    move.l  (sp)+,ListHandle(A5)
    ;set selflags to allow only one selection at a time!
    move.l  ListHandle(A5),A0
    move.l  (A0),A0
    move.b  #128,selfFlags(A0)
    ;put all the 'help' names (topics) into the list
    bsr    AddHelpNamesToList
    restoreregs
    rts

;
; <<<< Calculate cell width for list manager >>>>
;
Calc_Cell_Width
    bsr    Get_Rect_Of_Item
    move    DispRect+right(A5),D2
    move    DispRect+left(A5),D3
    sub     D3,D2
    sub     #16,D2          ;allow for scroll
    move    D2,cellwidth(A5) ;return value
    rts

;
; <<<< Add 'help' resource names to List >>>>
;
AddHelpNamesToList

```

```
;this is a routine to add all 'help' resource names
;to a list
```

```
saveregs
clr      -(sp)
move.l   #'help',-(sp)
_CountResources
move     (sp)+,D4 ;how many there are
move     D4,D7
beq      None_Here ;if no 'help' res's
                    ;here, quit!
sub       #1,D4 ;sub 1 for looping
                    ;(the DBRA later)
move     #false,-(sp) ;set drawing off
move.l   ListHandle(A5),-(sp)
_DoDraw
move.l   #nil,TheCell(A5) ;start with zero cell
;set ResLoad off (so resources are not
;automatically read into memory)
move     #false,-(sp)
_SetResLoad
```

```
AddhelpLoop
clr.l    -(sp)
move.l   #'help',-(sp)
move     D7,-(sp) ;res. id #
_GetResource
move.l   (sp),A4 ;leave handle on stack
                    ;for next routine
;get the resources name
pea      ResId(A5)
pea      ResType(A5)
pea      ResName(A5)
_GetResInfo
lea      ResName(A5),A3
clr.l    D3
move.b   (A3)+,D3 ;how many chars in
                    ;name for later
sub       #1,D7 ;decrement index #
                    ;D7 = the res id#
;d3 = how many characters in string,
;A3 = ptr to string data
;add a new row
clr      -(sp)
move     #1,-(sp)
move     #0,-(sp)
move.l   ListHandle(A5),-(sp)
_AddRow
move     (sp)+,D0
;set the cells data to the resources name
move.l   A3,-(sp) ;points to name
move     D3,-(sp) ;how many chars?
move.l   TheCell(A5),-(sp)
move.l   ListHandle(A5),-(sp)
_SetCell
dbra     D4,AddhelpLoop ;loop until D4 = 0
;set ResLoad back on
move     #true,-(sp)
_SetResLoad
move     #true,-(sp) ;turn drawing on
move.l   ListHandle(A5),-(sp)
_DoDraw
;update the list to show it
MOVE.L   DialogPtr(A5),A1
move.l   24(A1),-(sp)
move.l   ListHandle(A5),-(sp)
_Update
bsr      Frame_The_Rect
None_Here
restoreregs
rts
```

```
;
; <<<< Constants >>>>
;
```

```
DoneString dc.b 14,'Done with Help'
            .align 2
DoneViewing dc.b 12,'Done viewing'
            .align 2
```

```
;
; <<<< Global variables >>>>
;
```

```
WhichControl ds.l 1 ;used by _FindControl
TextHand ds.l 1 ;TE record Handle
ScrollHand ds.l 1 ;Handle of scroll bar
DialogPtr ds.l 1 ;Ptr to Help Dialog
LinesInWind ds 1 ;how many text lines fit in rect
AString ds.b 256 ;Space for a string
```

```
ListHandle ds.l 1 ;Handle to list
TheCell ds.l 1 ;Used by the List Mgr
ArrayRect ds.l 2 ;Rows & columns for List Mgr
cellwidth ds.w 1 ;width of cell in list
DataLength ds.w 1 ;Var used by _GetCell
```

```
ItemType ds.l 1 ;Vars used by _GetDitem
ItemHandle ds.l 1 ;Handle of Dialog Item
DispRect ds.l 2 ;Rect of Dialog Item
```

```
ResHand ds.l 1 ;Handle of a 'help' resource
ResId ds.w 1 ;Vars used by _GetResInfo ID#
ResType ds.l 1 ;res. type
ResName ds.b 256 ;res. name
```

```
ItemNumber ds.w 1 ;Item# returned by _ModalDialog
END
```

```
* File name: YourApp.R
* Example .R file for YourApp, which
* is an example application name
* that has been linked with DoHelp.
* Resource is also an example name, it
* should be the name of a resource file
* containing all of the needed resources for
* YourApp, including the 'help' resources for
* DoHelp which you can create using ResEdit
* This also creates the DLOG & DITL that
* DoHelp needs.
```

```
MDS1:YourApp
APPLTEST
```

```
Type DLOG
,2000
Help
56 32 318 476
visible NoGoAway
4
0
2000
```

```
Type DITL
,2000
3
```

```
Button
232 152 256 288
Done with help
```

```
userItem
8 8 200 434
```

```
staticText Disabled
208 96 224 344
Click on a Help topic (above) to view.
```

```
INCLUDE MDS2:YourApp.Code
INCLUDE MDS1:Resources
```

```
; File name:YourApp.Link
; Example Link file for
; DoHelp & YourApp
; YourApp is an example
; name, it should be
; whatever name
; you use for your
; applications code.
```

```
/Output YourApp.Code
/Type 'TEMP'
```

```
MDS2:YourApp
/Segment
MDS2:DoHelp
/Segment
```

```
$
```



Assembly Lab

Convert PICTs to Regions



Ted Cohn
San Jose, CA
Volume 5, Number 6

[Ted Cohn has been a long-time Mac enthusiast and used to be an Apple][hacker. He wrote a professional 6502 debugger for the Apple][called Bugbyter which Apple sold with its ProDOS assembly package. He graduated from UC Berkeley with a B.A. in Computer Science. Since then he has been working at Radius Inc., where he designed and wrote Tear-off Menus and a real-time magnifier for the Radius Two Page Display.]

Quickdraw Regions What's This All About?

Have you ever used a region? If so, have you ever tried creating irregularly-shaped ones? A year ago I wanted to write a desk accessory that would require several strangely-shaped regions. Unfortunately, I found no simple way to create them with the routines Quickdraw provides. Creating a region in Quickdraw is procedural: You must call `OpenRgn`, draw lines, rectangles, ovals, etc., and then call `CloseRgn` to define a region's boundary. This can be a tedious, almost unbearable process if you want to create bizarre regions by having to combine simple geometric shapes. My goal was to find a way to create regions in a non-procedural manner by drawing them with a standard paint program. Although Quickdraw does provide some useful functions like `UnionRgn`, `DiffRgn`, `SectRgn` and `XorRgn`, it does not provide a routine to convert a bitmap into a region – a routine I believe should be standard Quickdraw equipment.

The question arose: how do I turn a bitmap into a region? One could develop an algorithm to trace the edges of the bitmap and use Quickdraw to add each point individually to the region's boundary, but that seemed a bit primitive. The only way to find an efficient algorithm was to first understand the region structure itself. And as many already know, Apple has never divulged the region's structure – certainly cause for some fun detective work. I became sidetracked by this fascinating data structure.

The following explains the region format and my solution to the problem of region creation. Listed afterwards is `MakeRgn`, an assembly function to convert a bitmap into a Quickdraw region, and `Pict2Rgn`, an MPW tool which utilizes `MakeRgn` to convert 'PICT' resources to 'RGN' resources.

Let There Be Light!

This is what Apple thinks we ought to know about regions:

`rgnSize`: Integer;
`rgnBBox`: Rect;
`rgnData`: Mystery;

Inside Mac, page I-141, states, "A region can be concave or convex, can consist of one area or many disjoint areas, and can even have 'holes' in the middle." A region specifies exactly

which pixels lie inside and outside of a closed boundary. So, how does a region represent this boundary? Inside Mac describes only the header of the region structure, `rgnSize` and `rgnBBox`, which are not very interesting. `RgnSize` specifies the total number of bytes in the region data structure while `rgnBBox` contains the region's bounding rectangle. This is the smallest rectangle that will completely enclose the region, that is, each edge of the rectangle will touch at least one point of the region's boundary. It seems, however, that `rgnData` was left as an exercise to the developer – a complete mystery and a nagging curiosity.

Now let's get technical. It turns out that the `rgnData` structure is not as complex as it might seem. A region is like a bitmap in that it is arranged into scanlines and is read from left to right, top to bottom. This makes sense since regions are used to "mask" bitmaps which are also read from left to right, top to bottom. But instead of containing pixel data, the region need only contain *boundary information*.

Figure 1a is a doughnut-shaped picture we would like to convert into a region. Let's step through the conversion process. If we restrict ourselves to the horizontal dimension for the moment, we can isolate the boundary pixels of this bitmap by taking the exclusive-OR of itself with its *right-shifted* copy. Figure 1b is the result of this process. Notice that pixels which begin a horizontal line segment remain (pixel [5,0] for instance). The remaining pixels of each line segment are lost, but we gain an extra pixel at the end of the line segment. This additional pixel appears because of the XOR function and signifies the end of the segment. Therefore, in the horizontal dimension, two pixels are used to define the line segment.

Now we could use horizontal information alone to define the boundary, but we would have to store *every* scanline of the region. This would be extremely wasteful in cases where many contiguous lines are the same (as with large rectangular windows). Notice in Figure 1b that lines three and four are equal, lines six through eight are equal, and lines ten through eleven are equal.

Redundant information in the vertical dimension is eliminated by taking an additional XOR of the bitmap in Figure 1b, except, we take the XOR of the bitmap with its *down-shifted* copy. Each scanline is replaced by an XOR of itself with the previous line. Figure 1c shows the result of this second XOR. Notice that lines four, seven, eight and eleven are now blank. This means that only the first line of a run of equal lines is necessary to define them all.

The Quickdraw region is basically a list of the points (represented by black pixels) in Figure 1c. These points are called "inversion points." Why? Well, if we were to read a scanline from left to right, they would essentially *invert* the state

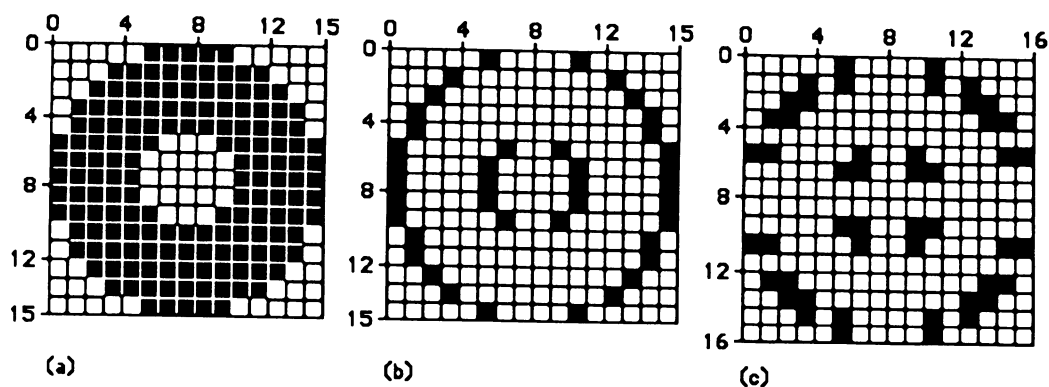


Figure 1. Three stages of finding a picture's inversion points. (a) Source bitmap where black pixels are inside the region. (b) Horizontal XOR with right-shifted copy. (c) Final bitmap resulting from horizontal and vertical XORs with inversion points ready to be converted to region format.

of being inside or outside of the region. Unfortunately, this definition is not completely accurate. There is a little more to inversion points than meets the eye. One must realize that no single line of inversion points (except for the first line) can reliably specify which pixels lie inside or outside of a particular line's boundary. Because the region is not a random access data structure like a bitmap, it must be "played back," line by line, starting from the top. The first line of inversion points defines the line segment(s) of the top line, but inversion points of successive lines do not directly inform us of the region's boundary. Boundary information provided by inversion points trickles down each line as the region is processed. This boundary information comes in the form of horizontal "markings" which are set and cleared by inversion points. An inversion point on one line can cancel out a mark made by an inversion point from a previous line.

To illustrate, let's unravel the boundary of the second line of Figure 1c. The line segment defined on the first line is assumed to be the same for the second line, so marks [5] and [10], which were set, carried through. The second line contains four inversion points: [3,1], [5,1], [10,1] and [12,1]. Inversion point [3,1] defines the new *starting point* of the line segment. Point [5,1] cancels mark [5] made by the previous inversion point [5,0] so the line segment will not end at point [5,1]. The line segment might also have ended at point [10,1] if it were not for the inversion point [10,1] telling us to cancel mark [10]. The line segment then ends at point [12,1]. For line three, only two marks carry through: [3] and [12]. The other two were canceled out.

During runtime, these marks are stored in the form of an internalized bitmapped line. The bitmapped line is modified from the inversion points that are read in and can be conveniently used to plot the bits of the region on the screen or mask a bitmap in CopyBits.

At this point, let me spell out the rgnData structure in detail. RgnData is a list of two or more scanlines. Each scanline begins with its vertical coordinate. (Every coordinate in the structure is a word in length.) Inversion points are then listed from left to right by entering their horizontal coordinates one after another. Note that there are always an even number of inversion points per scanline because the region is closed. An end mark, \$7FFF,

terminates the scanline. The remaining scanlines come one after another, each starting with its vertical coordinate. When there are no more scanlines remaining, a final end mark, \$7FFF, terminates the rgnData structure. Figure 2 shows the entire Quickdraw region data corresponding to the inversion points of Figure 1c.

MakeRgn Joins the Quickdraw Family

Since it is desirable to draw a region with a paint program, we define the region in bitmap form such that all black pixels in a bitmap are logically included in the region and all white pixels are logically excluded. MakeRgn (Listing 1) follows the process outlined above in converting this bitmap to Quickdraw region format. MakeRgn first does a horizontal XOR with its righted shifted copy and then does a vertical XOR with its down-shifted copy. We could take the vertical XOR first; the order doesn't matter. This produces a bitmap with inversion points whose coordinates are entered into the rgnData structure. MakeRgn will create and return the region handle, so don't call NewRgn first. If MakeRgn is provided with an empty bitmap, it will return an empty region. It does not handle Memory Manager errors. If you are concerned about heap space, you can test to see if there exists a free block which is about one hundred bytes larger than your source bitmap. The rgnSize and rgnBBox fields will be calculated and filled in too. The coordinates of the rgnBBox will reflect the bitmap's bounds rectangle and the location of the region within it.

Tools of the Trade

Pict2Rgn, an MPW Tool shown in Listing 2, lets one convert 'PICT' resources into 'RGN' resources. Listing 3 is the make file associated with Pict2Rgn. To use it, place 'PICT' resources into a resource file and execute Pict2Rgn with that file name. It will convert each picture and add its corresponding 'RGN' resource to that file with the same ID and name as the source 'PICT'. The program does not check to see if the resource file already contains regions with the same ID's, so make sure it is clear of 'RGN' resources before converting pictures. Pict2Rgn demonstrates how to open a GrafPort and draw a picture into that port. Once drawn, the conversion to region format is done by calling MakeRgn with a pointer to the GrafPort's bitmap.

A nice extension to this tool might be a Region Editor that would allow one to graphically manipulate regions and apply Quickdraw functions to them. Another would be a Control Editor to let one create interesting new controls for dialog boxes. The basic method outlined in Pict2Rgn should be helpful in writing your own utility or package to use MakeRgn.

rgnSize	009C
rgnBBox	0000 0000 000F 000F
rgnData	0000 0005 000A 7FFF 0001 0003 0005 000A 000C 7FFF 0002 0002 0003 000C 000D 7FFF 0003 0001 0002 000D 000E 7FFF 0005 0000 0001 0006 0009 000E 000F 7FFF 0006 0005 0006 0009 000A 7FFF 0009 0005 0006 0009 000A 7FFF 000A 0000 0001 0006 0009 000E 000F 7FFF 000C 0001 0002 000D 000E 7FFF 000D 0002 0003 000C 000D 7FFF 000E 0003 0005 000A 000C 7FFF 000F 0005 000A 7FFF 7FFF

Figure 2. Region data corresponding to bitmap in Figure 1c. The region is 156 bytes long. The rgnBBox rectangle is (0,0,15,15). RgnData lists each scanline of inversion points which begin with the vertical coordinate and end with the constant \$7FFF.

A Tidbit: What a Drag

If you are interested in dragging outlines of regions, there is a faster way than calling FrameRgn continuously. First, make a copy of the region, then InsetRgn(theRgnCopy, 1, 1) and DiffRgn(theRgn, theRgnCopy, theRgn) to create a region outline. Then call PaintRgn(theRgn) to drag the region outline around. FrameRgn is slow because it performs this same process every time it is called.

Any Alternatives?

Not surprisingly, bitmaps can be used to represent boundaries too. Like our definition above, we can say that black pixels in a bitmap represent the interior of the boundary. Depending on the complexity of the boundary, a bitmap can be considerably smaller in size than its equivalent Quickdraw region. The region data in Figure 2 is 156 bytes long, whereas the bitmap in Figure 1 is only 30 bytes long! This is because each inversion point requires entire word of storage. If there is more than approximately one inversion point per word of pixel data, then it is more space efficient to use a bitmap to represent the region. Still, Quickdraw regions are perfect for use by the Window Manager because the regions manipulated are usually rectangular and generally large requiring at most several hundred bytes. (Imagine Quickdraw using entire bitmaps just for region manipulation!) Thus, the larger, less complex the region, the more space efficient it is compared to its bitmap equivalent. On an historical note, region data was originally compressed by Quickdraw. Regions were generally around one third the size they would be today, but region manipulation was slower because of the on-the-fly compression/decompression. Region compression was removed before shipping the 128K Mac. While I don't prefer a larger region structure, I think Apple was wise in opting for speed in the windowing environment - especially now that we have megabyte systems instead of 128K Macs to contend with.

Region Wrap-up

I hope this utility helps promote the creation of regions for new controls and windows. While user interface standards are important, I think a little variation wouldn't hurt. As to Apple's

changing the region format in the future, I have this to say: It is easier to design regions non-procedurally than procedurally. If the format changes, another routine can be easily created to convert a bitmap into a region. In the future, I hope Apple will add a similar conversion routine to its graphics repertoire. If you use MakeRgn, I would be most interested in hearing about it. My AppleLink address is D0959. Happy Macing!

Listing 1. MakeRgn assembly code.

```

;
; MakeRgn.a
;
; Written by Ted Cohn, March 1987.
; Copyright 1988 By Ted Cohn.
;
; MakeRgn converts a bitmap image into a Quickdraw region.
; Black pixels are logically included in the resultant region
; whereas
; white pixels are not included. No special conditions
; required
; of the bitmap - it may be arbitrarily complex. Empty bitmap
; will
; simply yield an empty region. rgnBBox of resultant rgn may
; be smaller than initial bitmap bounds if image does not use
; full extent of the bitmap. rgnBBox will be smallest rect
; enclosing the resultant region.
;
; Algorithm Summary:
;
; The idea behind the conversion is simple. The key is in
; finding
; inversion points of the picture which define the region. A
; source bitmap is first XOR'd with its right-shifted copy.
; resultant bitmap is then XOR'd with its down-shifted copy.
; produces a set of inversion points which define the region
; boundary. next step is to take these points within final
; bitmap and convert them to Quickdraw region format.
;
; FUNCTION MakeRgn(srcMap: Bitmap): RgnHandle;
;
; Modification History:
;
; 25 Mar 87    New Today.
; 26 Mar 87    Ironed out those bugs!
; 30 Mar 87    Changed to a Pascal function.
; 28 Mar 88    Optimized and reduced code.
;
;
; MACHINE MC68000
;
; INCLUDE 'Traps.a'
; INCLUDE 'QuickEqu.a'
;
; ENDMARK EQU $7FFF ; end of rgn/line mark [word].
;
; MakeRgn's stack frame:
;
; MRFrame RECORD (A6Link),DECR
; dstRgn DS.L 1 ; output region [RgnHandle].
; srcMap DS.L 1 ; source bitmap [Ptr].
; Return DS.L 1 ; return address [Ptr].
; A6Link DS.L 1 ; old A6 value [long].
; srcRect DS.B 8 ; temporary [Rect].
; dstRect DS.B 8 ; temporary [Rect].
; rgnMap DS.B 14 ; temp bitmap [BitMap].
; VarSize EQU * ; size of local variables.
; ENDR
;
; MakeRgn PROC EXPORT
; WITH MRFrame
; LINK A6,#VarSize; create local stack frame.

```

```

MOVEM.L A1-A3/D0-D7,-(SP) ; save working registers.
MOVE.L srcMap(A6),A3 ; load ptr to source bitmap.
MOVEQ #0,D0 ; clear for multiplying.
MOVEQ #0,D4 ; use D4 for faster clears.
MOVE.W rowBytes(A3),D7 ; load srcMap rowBytes.
; Find number of lines in source bitmap. The rgnMap will have
; the same number of lines plus one and two more rowbytes.
MOVE.W bounds+bottom(A3),D0 ; load srcMap bottom.
SUB.W bounds+top(A3),D0 ; height = bottom-top.
ADDQ.W #1,D0 ; height++.
ADDQ.W #2,D7 ; rowBytes += 2.
; Fill in rgnMap rowBytes and baseAddr fields. Allocate
; memory for the temporary rgnMap image buffer.
MOVE.W D7,rgnMap+rowBytes(A6); store new rowBytes.
MULU D7,D0 ; calculate bitmap size.
_NewPtr ,clear ; create rgnMap image buffer.
MOVE.L A0,rgnMap+baseAddr(A6); store start of buffer.
; rgnMap's bounds will be one pixel larger to the right and
; bottom.
MOVE.L bounds+topLeft(A3),D0 ; get original bitmap bounds.
MOVE.L bounds+botRight(A3),D1
ADDQ.W #1,D1 ; right++.
MOVE.L D0,rgnMap+bounds+topLeft(A6)
MOVE.L D1,rgnMap+bounds+botRight(A6)
ADDQ.W #1,rgnMap+bounds+bottom(A6)
; Now make right-shifted copy in rgnMap.
ADDQ.W #1,D0 ; left++.
MOVE.L D0,dstRect+topLeft(A6); dstRect is right-shifted
one.
MOVE.L D1,dstRect+botRight(A6)
MOVE.L A3,-(SP); srcBits = srcMap.
PEA rgnMap(A6); dstBits = rgnMap.
PEA bounds(A3); srcRect = srcMap.bounds.
PEA dstRect(A6); dstRect.
MOVE.W D4,-(SP); srcCopy mode.
MOVE.L D4,-(SP); no maskRgn.
_CopyBits
; XOR srcMap with right-shifted copy rgnMap & store in rgnMap.
MOVE.L A3,-(SP); srcBits = srcMap.
PEA rgnMap(A6); dstBits = rgnMap.
PEA bounds(A3); srcRect = srcMap.bounds.
PEA bounds(A3); dstRect = srcMap.bounds.
MOVE.W #srcXor,-(SP); srcXor mode.
MOVE.L D4,-(SP); no maskRgn.
_CopyBits
; XOR rgnMap with down-shifted copy of rgnMap.
MOVE.L rgnMap+bounds+topLeft(A6),srcRect+topLeft(A6)
MOVE.L rgnMap+bounds+botRight(A6),srcRect+botRight(A6)
MOVE.L srcRect+topLeft(A6),dstRect+topLeft(A6)
MOVE.L srcRect+botRight(A6),dstRect+botRight(A6)
SUBQ.W #1,srcRect+bottom(A6)
ADDQ.W #1,dstRect+top(A6)
PEA rgnMap(A6); srcBits = rgnMap.
PEA rgnMap(A6); dstBits = rgnMap.
PEA srcRect(A6); srcRect = top rectangle.
PEA dstRect(A6); dstRect = down-shift one.
MOVE.W #srcXor,-(SP); srcCopy mode.
MOVE.L D4,-(SP); no maskRgn.
_CopyBits
; We've exposed the inversion points of the picture. Time to
; count the number of rows and black pixels in the rgnMap to

```

```

determine
; the size of the RgnHandle we will soon fill in.
;
MOVE.L rgnMap+baseAddr(A6),A0; start at topLeft of bitmap.
LSR.W #1,D7 ; rowWords = rowBytes/2.
SUBQ.W #1,D7 ; loop rowWords times.
MOVE.W rgnMap+bounds+top(A6),D1; current Y coordinate.
MOVE.W rgnMap+bounds+bottom(A6),A1
; last Y coord. for end test.
MOVEQ #0,D5 ; row count = 0.
Count
MOVE.W D7,D6 ; x loop on rowWords.
MOVEQ #0,D3 ; clear line flag.
@0
MOVE.W (A0)+,D0 ; load next rgnMap word.
BEQ.S @2; skip blank words.
TST.B D3; is line flag already set?
BNE.S @1; yes -> skip.
STD3 ; no -> set line flag.
ADDQ.W #1,D5 ; row count++.
@1
ADDQ.W #1,D4 ; add number of bits
MOVE.W D0,D2 ; in rgnMap word to
SUBQ.W #1,D2 ; total bit count.
AND.W D2,D0
BNE.S @1
@2
DBRA D6,@0 ; loop if more words to read.
ADDQ.W #1,D1 ; y++.
CMP.W D1,A1 ; y <= bottom?
BGT.S Count ; yes -> loop.
; Determine size of new RgnHandle.
TST.W D5; was the rgnMap empty?
BNE.S BuildRgn ; no -> skip.
SUBQ.W #4,SP ; yes -> get new empty region.
_NewRgn
MOVE.L (SP)+,dstRgn(A6) ; stuff result.
BRA Done; exit.
BuildRgn
; Calculate number of bytes to allocate for output region
handle.
MOVEQ #12,D0 ; (12 for header & end mark.)
ADD.W D1,D1 ; (4*number of lines in map for
ADD.W D1,D1 ; each Y coordinate and
ADD.W D1,D0 ; end mark.)
ADD.W D4,D4 ; (2*number of inversion
ADD.W D4,D0 ; points in the entire map.)
MOVE.W D0,D4 ; remember the region size.
_NewHandle ,clear ; allocate a cleared RgnHandle.
MOVE.L A0,dstRgn(A6) ; store handle in result.
; Translate rgnMap (containing inversion points) into Quick-
; draw Region
; format. We do not need to lock block because we will be
; making no
; trap calls which might rearrange the heap. DstRect will be
; used to
; help us find smallest enclosing rectangle for the region.
MOVE.L (A0),A0 ; point to rgn data.
MOVE.W D4,rgnSize(A0) ; store region size.
MOVE.L A0,A2 ; save ptr for the end.
LEA rgnData(A0),A0 ; offset ptr to rgnData.
MOVE.L rgnMap+baseAddr(A6),A1; point to topLeft of bitmap.
MOVE.W rgnMap+bounds+top(A6),D1; y = top.
MOVE.W D1,dstRect+bottom(A6); initialize dstRect...
MOVE.W rgnMap+bounds+left(A6),dstRect+right(A6)
MOVE.L rgnMap+bounds+botRight(A6),dstRect+topLeft(A6)
; The line flag tells us if the scanline is not completely

```

```

blank after
; reading whole line. If not blank, we add end-of-line mark.
;
Translate
    MOVE.W    D7,D6    ; load word count.
    MOVEQ     #0,D5    ; clear line flag.
    MOVE.W    rgnMap+bounds+left(A6),D3
; x = left edge.
@0
    MOVE.W    (A1)+,D0 ; load srcMap word.
    BNE.S     @NotEmpty ; only process non-zero words.
    ADD.W     #16,D3    ; skip zero words for speed.
    BRA.S     @4
@NotEmpty
    TST.B     D5; is line flag set?
    BNE.S     @1; yes -> skip.
    STD5      ; no -> set line flag.
    MOVE.W    D1,(A0)+ ; store y coord. in structure.
    CMP.W     dstRect+top(A6),D1 ; now find the topmost
    BGE.S     @MaxBottom ; row of the region.
    MOVE.W    D1,dstRect+top(A6) ; top = min(top,y).
@MaxBottom
    CMP.W     dstRect+bottom(A6),D1 ; now find the bottommost
    BLE.S     @1; row of the region.
    MOVE.W    D1,dstRect+bottom(A6) ; bottom = max(bottom,y).
@1
    OR.$10,CCR ; set the X flag.
    ADDX.W    D0,D0    ; shift in end-bit-marker.
@2
    BCC.S     @3; branch if MSB clear.
    MOVE.W    D3,(A0)+ ; store x coord. in structure.
    CMP.W     dstRect+left(A6),D3 ; now find the leftmost
    BGE.S     @MaxRight ; edge of the region.
    MOVE.W    D3,dstRect+left(A6) ; left = min(left,x).
@MaxRight
    CMP.W     dstRect+right(A6),D3 ; now find the rightmost
    BLE.S     @3; edge of the region.
    MOVE.W    D3,dstRect+right(A6) ; right = max(right,x).
@3
    ADDQ.W    #1,D3    ; increment the x coordinate.
    ADD.W     D0,D0    ; shift msb bit into carry.
    BNE.S     @2; repeat if more one-bits.
@4
    DBRA      D6,@0    ; loop on rowWords-1.
    TST.B     D5; empty line?
    BEQ.S     @5; yes ->
    MOVE.W    #ENDMARK,(A0)+ ; no, store end-of-line mark.
@5
    ADDQ.W    #1,D1    ; increment the y coordinate.
    CMP.W     rgnMap+bounds+bottom(A6),D1
; loop if y <= bottom rgnMap.
    BLT.S     Translate
;
; We are finished entering information into the rgnData field.
; Mark the end of the region and fill in the rgnBBox field.
;
    MOVE.W    #ENDMARK,(A0) ; store end-of-region mark.
    MOVE.L    dstRect+topLeft(A6),rgnBBox+topLeft(A2)
    MOVE.L    dstRect+botRight(A6),rgnBBox+botRight(A2)
Done
    MOVE.L    rgnMap+baseAddr(A6),A0 ; deallocate temporary
bitmap.
    _DisposPtr
    MOVEM.L   (SP)+,A1-A3/D0-D7 ; restore working registers.
    UNLK      A6; unlink the stack frame.
    MOVE.L    (SP)+,A0 ; load return address.
    ADDQ      #4,SP ; pop parameter.
    JMP       (A0); return.
ENDWITH
ENDPROC
END

```

Listing 2. Pict2Rgn.

```

/*
NAME
    Pict2Rgn - convert PICT resources to Quickdraw region
    format.
SYNOPSIS
    Pict2Rgn [file] [-v]

    file = any resource file containing PICT resources
    -v = verbose flag
DESCRIPTION
    Converts PICT resources within a file to Quickdraw
    region format. ID's of the generated regions will be the
    same as the source PICT's. region resource type is 'RGN'.
    Does not check to see if RGN resources are already there.
AUTHOR
    Written Ted Cohn, March 1988.
    Copyright Ted Cohn, 1988.
*/

#include <types.h>
#include <stdio.h>
#include <resources.h>
#include <quickdraw.h>
#include <memory.h>

#define RgnType 'RGN'

pascal RgnHandle MakeRgn(srcMap)
    BitMap *srcMap;
    extern;

main(argc, argv)
    int argc;
    char *argv[];
{
    Rect frame; /* picture frame */
    GrafPort gp; /* offscreen GrafPort pictures are drawn */
    short index; /* for-loop variable */
    short refNum; /* the resource file's refNum */
    short pictCount;
    /* number of picts in resource file to convert */
    Handle pictHandle; /* handle to current picture */
    short rowWords; /* #of words across picture's bitmap */
    BitMap srcMap; /* GrafPort's bitmap */
    short theID; /* resource ID from GetResInfo */
    ResType theType; /* resource TYPE from GetResInfo */
    RgnHandle theRgn; /* hdl region converted from picture */
    Str255 theName; /* resource NAME from GetResInfo */
    short verbose; /* boolean signals to print more info */

    verbose = (argc>2); /*boolean more than 2 arguments given */

    /* Must initialize Quickdraw for the conversion process */
    InitGraf(&gd.thePort);
    /* if no argument supplied, then print info about tool */
    printf("\n");
    if (argc<2) {
        fprintf(stderr, "\nUsage: Pict2Rgn <resource file>");
        fprintf(stderr, "\n -Converts all PICT resources to
        Quickdraw region format.");
        fprintf(stderr, "\n -ResType is 'RGN', ID's correspond to
        PICT ID's.");
        fprintf(stderr, "\n -Written by Ted Cohn, March,
        1988.\n");
    }
    exit(1);
    /* Try to open resource. If errors, report and exit. */
    if ((refNum = OpenResFile(argv[1])) == -1) {
        fprintf(stderr, "\nCan't open the resource file %s.\n",
        argv[1]);
    }
    exit(1);
}

```

```

/* Find out how many to convert. Exit if no pics in file.*/
if ((pictCount = Count1Resources('PICT')) == 0) {
    fprintf(stderr, "\nThere are no PICTs in this file to
convert.\n");
    CloseResFile(refNum);
    exit(1);
}

if (verbose)
    fprintf(stderr, "PICT Count = %d\n", pictCount);
/* Now convert each PICT to RGN format */
for (index = 1; index <= pictCount; index++) {
    /* Get a picture and lock it down */
    HLock(pictHandle = Get1IndResource('PICT', index));
    /* Must get the picture's ID */
    GetResInfo(pictHandle, &theID, &theType, &theName);
    /* Get the picture's frame rectangle */
    frame = *((Rect *) (*pictHandle + 2));
    /* Calculate size of srcMap we need to allocate */
    rowWords = (frame.right-frame.left-1)/16+1;
    srcMap.baseAddr = NewPtr((long) ((frame.bottom-
frame.top)*rowWords*2));
    srcMap.rowBytes = rowWords*2;
    srcMap.bounds = frame;

    /* Open offscreen GrafPort and set bitmap to our srcMap */

    OpenPort(&gp);
    SetPortBits(&srcMap);
    RectRgn(gp.clipRgn, &frame);
    RectRgn(gp.visRgn, &frame);
    gp.portRect = frame;
    DrawPicture((PicHandle) pictHandle, &frame);
    ClosePort(&gp);
    /* Now convert the picture from a srcMap to a region */
    theRgn = MakeRgn(&srcMap);
    ReleaseResource(pictHandle);

```

```

    DisposPtr(srcMap.baseAddr);
    if (verbose)
        fprintf(stderr, "Converted (%d) PICT ID = %d\n", index,
theID);

    /* Add the new region to the file and dispose of it */
    AddResource(theRgn, RgnType, theID, &theName);
    WriteResource(theRgn);
    ReleaseResource(theRgn);
}
/* All done, so close the resource file */
CloseResFile(refNum);
exit(0);
}

```

Listing 3. Make file for Pict2Rgn.

```

# File:    Pict2Rgn.make
# Target:  Pict2Rgn
# Sources: MakeRgn.a Pict2Rgn.c
# Created: Sunday, April 3, 1988 11:25:18 AM

```

```

MakeRgn.a.o f Pict2Rgn.make MakeRgn.a
Asm MakeRgn.a
Pict2Rgn.c.o f Pict2Rgn.make Pict2Rgn.c
C Pict2Rgn.c
Pict2Rgn ff Pict2Rgn.make MakeRgn.a.o Pict2Rgn.c.o
Link -t MPST -c 'MPS ' 0
    "(Libraries)"CRuntime.o 0
    "(Libraries)"StdCLib.o 0
    "(Libraries)"CSANELib.o 0
    "(Libraries)"CInterface.o 0
MakeRgn.a.o 0
Pict2Rgn.c.o 0
-o Pict2Rgn

```



Developer Notes

MDS->MPW



Dan Weston
Portland, OR
Volume 5, Number 6

Converting MDS files to MPW

This article describes how to convert MDS assembler source files to the MPW environment. I don't go into all the details, but try and give you enough of the major points so that you can get your old files over to MPW and get more productive. MPW is a very rich and wonderful environment and I am purposely ignoring many of its subtle and not-so-subtle features in this article for the sake of clarity (and deadline).

Declare a MAIN Procedure

MPW expects you to declare at least one code module in your source file. There are several ways to start a code module, but the easiest is to put the MAIN directive on the same line at the label at the start of your code producing statements. For example, the following code fragment establishes the label "Start" and the main code module.

```
Start    MAIN          ; code statements start here
```

If you don't declare a code module, either by using MAIN or PROC or FUNC, MPW will complain heartily.

Declare Global Data Record

In MDS, it is customary to declare global variables with the DS directive. The MDS assembler gathers all DS directives together into a common global data block and tells the linker to locate the data block at -256(a5) when the program is loaded. (This location is the default and you can change it with Link directives, although most people don't bother to change it. Later in this article when we discuss the QuickDraw globals, you will see that the location of the application globals in MDS is important.) All the labels associated with the DS statements become negative offsets from A5 to the proper location in the data area. For example, if you have the following data statements in your MDS file:

```
thePoint    ds.l      1      ;space for a Point
theRect     ds.w      4      ;4 words for a Rect
```

You can access those variables by using the name of the variable as an offset from register a5, as in this line of code.

```
move.l      thePoint(a5),-(sp)
```

In MDS files, DS directives can be placed anywhere in the file, and the assembler will gather them all together and make a single data block. The MPW assembler also gathers up DS statements and makes code modules and the MPW linker com-

bines all the data modules and place them just below the a5 boundary points when the program is loaded. The MPW linker does not give you an automatic 256 byte cushion below the a5 boundary the way the MDS linker does. Labels for DS statements become negative offsets from register a5, however, in much the same way as in MDS. You can access MPW assembler data variables by using the variable name (label) as as offset from register a5, as shown in the previous example.

A better way to declare global variables in MPW is to gather all your DS statements together into a RECORD and then use that data template to access all your globals as fields of the template. For example, to convert the global variables from the MDS example above, you would place the following data declaration at the beginning of your file (after the 'INCLUDES' and 'EQU' statements but before the first code statements).

```
MyData      RECORD
thePoint     ds.l      1
theRect      ds.w      4
ENDR
```

Once you declare a block of globals this way, you can access the individual variables without referencing register a5, since the MPW assembler will automatically generate the ar reference for you. You access a variable by using its data module name (eg. 'MyData') and a field name from the data template (eg. 'thePoint'), much like a record structure in C or Pascal. For example, you can access the variable at the label 'thePoint' with the following code.

```
move.l      MyData.thePoint,-(sp)
```

You can also declare implicit data modules by using the WITH directive. For example, if you preceded the previous example with the statement:

```
Start      MAIN
           WITH      MyData
           ; Code statements begin here
```

you could dispense with the data module name when accessing individual fields of the data module, so that the access statement becomes:

```
move.l      thePoint,-(sp)
```

In summary then; when converting an MDS file to MPW, gather all your DS directive lines into a single data module and give the data module name (the label that appears on the same line as the RECORD directive) in a WITH statement before accessing any of the variables in the data module. You can then access the

variables by using their names without worrying about referencing register a5.

Declare QuickDraw Globals

One of the hidden features of the MDS linker is that it places the application globals 256 bytes below the a5 boundary by default. The MPW linker places the application globals right below the a5 boundary, without any cushion. Most casual assembly language programmers (is there such a thing?) just accept the default and don't really think about it, but it can get you in a lot of trouble when converting between MDS and MPW if you don't take this difference into account.

In particular, it is common for MDS programs to assume that the QuickDraw globals will be placed just below the a5 boundary, extending downward in memory for 206 bytes. The ROM call InitGraf expects a pointer to the 203rd byte of a 206 byte area for the QuickDraw globals. Because MDS gives a 256 byte cushion below the a5 boundary before the application's globals, MDS programs typically call InitGraf with a pointer to a long word four bytes below the a5 boundary, as shown below.

```
; typical MDS technique
    pea    -4(a5)
    _InitGraf
```

This works great in MDS programs because the QuickDraw globals grow downward from the a5 boundary but are not large enough to outgrow the 256 byte cushion provided by the MDS linker.

In MPW, you are not so lucky. You must explicitly allocate some global storage for the Quickdraw globals in a data module and then pass a pointer to that module when calling InitGraf. If you don't do this and use the previous code to initialize QuickDraw, your globals and the QuickDraw globals will share overlapping data space beneath the a5 boundary, and that will cause your program to crash as your program and QuickDraw write over each others globals.

The sample program provided with MPW shows a good method for allocating data space for the QuickDraw globals. Listed below is a simplified version of that method. First, declare a data module for the QD globals. This declaration should come just before or just after your application's data declaration, but before any code statements, as shown below.

```
QDGlobals    RECORD    ,DECREMENT
thePort      DS.L      1
white        DS.B      8
black        DS.B      8
gray         DS.B      8
ltGray       DS.B      8
dkGray       DS.B      8
arrow        DS.B      cursRec
screenBits   DS.B      bitmapRec
randSeed     DS.L      1
ORG          -grafSize
ENDR
```

Once the QD globals data module is declared, it will be combined with any other data modules in your program and

loaded just below the a5 boundary when the program is loaded. Because you don't know exactly where the QD globals will be loaded in relation to the other data modules (alright, if you dig through the MPW manuals you can probably figure out how the data modules are ordered).

In order to use the QuickDraw globals that are defined as a data module, you should place the name of the data module in the WITH directive along with the name of your application's data module, as shown below.

```
Start      MAIN
WITH      MyData,QDGlobals

; Code statements begin here
```

And, once you are using the QD globals you have defined, you can point directly to the QD data area when initializing QuickDraw, as shown below.

```
; thePort is a variable in QDGlobals
    pea    thePort
    _InitGraf
```

By explicitly defining a data area for the QuickDraw globals and pointing to it when initializing QuickDraw you can avoid the data overlap that occurs when you convert an MDS program without thinking about the details.

Misc Details

— MPW uses IMPORT and EXPORT instead of the XDEF and XREF directives found in MDS.

— Macros are defined differently in MPW and MDS. For example, take a look at the following macro defined for both MDS and MPW:

MDS macro:

```
MACRO    _SFGetFile =
MOVE.W   #2,-(SP)
_Pack3
|
```

MPW equivalent macro:

```
MACRO
_SFGetFile
MOVE.W   #2,-(SP)
_Pack3
ENDM
```

— The labels used with EQU and SET statements in MPW must be in the first column, unlike MDS.

— RMaker and rez source files are totally incompatible, but you use your old resource files by either "including" them in your rez file or running derez on them to get a rez compatible source file.

— Make files made by the "Create Build Commands..." menu option in MPW typically are a good way to build your programs, but you can remove the link files "Runtime.o" and "Interface.o" put in the make file by default.



Assembly Lab

Serial (Serial) I/O



Frank Henriquez
Los Angeles, CA
Volume 5, Number 7

Programming The Serial Ports

The Macintosh serial ports are a neglected programming subject. Few (if any) books on programming cover them, and this lack of information is compounded by an absence of source code to study.

The simplest way of programming the serial ports is to use a high level language and fool the compiler into thinking that the ports are disk files. Then you can use the language's generic file I/O routines for serial I/O. While this solution provides a simple way of dealing with the serial ports, it may not provide the speed or flexibility that may be required in certain applications.

I thought that a better approach would be to write a set of low level routines that can be linked with compiler generated code, or with assembly language. How difficult could this be? All I'd need was the hardware manual for the Mac's serial port chip, a copy of Inside Macintosh, a steady source of caffeine and a few hours of coding...right?

I quickly discovered that the task of writing drivers for the Mac is serious business. Life is too short for the hassles involved in trying to write a driver for the the complex (and bizarre) serial port chip, a Zilog 8530 SCC. Besides, it's already been done; Kirk Austin's excellent articles "A Midi Library for Pascal" and "Midi Demo Uses Midi Library" in the July, 1987 and December, 1987 issues of MacTutor show how to write drivers for this chip, if you're interested in working close to the bare metal of the machine.

How I Learned To Stop Complaining And Love The Device Manager

or

"Gee, I didn't realize it would be this easy"

Midi has certain timing and handshaking constraints that demand specialized serial drivers. I needed something a lot simpler; a set of routines that could send and receive data, at baud rates that would probably never exceed 19200 kilobaud. There had to be a way of doing this that didn't involve writing a complex driver. As often happens when programming the Macintosh, a solution became obvious only after re-reading several chapters of Inside Macintosh (or to put it another way: "when all else fails, read the manual"). While the Serial Drivers chapter would seem like the obvious place to look for information on using the serial ports, the Device Manager chapter is far more important; the Macintosh treats the serial ports as devices, so accessing the drivers requires using the Device Manager I/O routines.

The Device Manager can be divided into Pascal based file I/O routines, and low level traps. The high level Pascal routines (they can also be called from C. But you already knew that) have an extremely simple calling sequence, and insulate the program-

mer from the inner workings of the Device Manager. However, since these routines are not in the ROMs, the glue routines have to be linked with any assembly language program that uses them.

The low level Device Manager traps require more work to use, but they provide greater control over the I/O. I could have used the high level routines in my program, but since I had to use a low level Device Manager trap to configure the serial port, I decided to write everything using the other low level traps; the code is smaller and probably a bit faster.

The serial ports have been assigned four driver reference numbers, which are used by the Device Manager to identify the device it is working with:

Modem port	input	(port A)	-6
Modem port	output	(port A)	-7
Printer port	input	(port B)	-8
Printer port	output	(port B)	-9

I've used these reference numbers directly in the code, instead of using the reference numbers returned by the _Open trap (the numbers *should* be the same). It simplifies the coding, at the risk of making the routines non-functional if and when Apple decides to change the reference numbers.

The Device Manager traps use a block of memory (pointed at by register A0) called a **parameter block**. It holds the pointers, commands and flags that may be required by the Device Manager trap being called. Depending on the call, it may also contain information returned from a device. A generic parameter block can be up to 80 bytes long, but when calling the serial ports most of the parameter block entries can be ignored. This still means that my routines have to set up and maintain a parameter block each time they are called. At first, even this small overhead in execution time and memory requirements made me uncomfortable; however this is an acceptable compromise considering the amount of code required to write a low level serial driver.

In retrospect, using the Device Manager to control the serial ports turned out to be an easy task, although I spent more time than I'm willing to admit learning how to write these serial I/O routines. In the process, I discovered that Apple provides serial port drivers in a resource called SERD (built into the ROMs of the more recent Macs). At the time I wrote my routines, SERD was not in ROM and its use was (and still is) poorly documented. It was also more code than I had expected, so I went ahead with my routines.

One of my first attempts at using the Device Manager was a simple terminal program written in Turbo Pascal. It worked, but I decided to re-write it in assembly because (I know many of you will cringe) I *enjoy* writing in assembly language! I realize that it's an acquired taste and that most of you would prefer to keep

your contacts with assembly language to a minimum, so these routines have been written to make them accessible from Pascal. The routines will also work with C, but you may have to modify them to match the C parameter passing protocol. I've also written a small terminal program that shows the routines in use. The bulk of the program consists of code to use TextEdit, manipulate the clipboard and code to do the usual housekeeping routines. You'll almost need a magnifying glass to find the calls to the serial routines!

Serial I/O Routines

The serial I/O routines make extensive use of the stack, for parameter passing and for holding the parameter block required by the Device Manager routines they'll be calling.

Parameter blocks are common structures, and can be found in most Macintosh I/O routines. In assembly language, we can define a parameter block as a buffer in the global variable space, or we can use the approach used by many high level language compilers and define the parameter block as a temporary structure on the stack. On entry, each routine sets aside a small section of the stack, known as a stack frame, and points to it with register A6. This private section of memory is then set aside as a parameter block.

Making the parameter block a temporary structure on the stack saves us from having to declare it as part of the global variable space; the local stack frame and all its contents are discarded when the routine terminates. Using this technique increases the complexity of the routines, but their portability and ease of use far outweighs the minor increase in execution time and code size.

Before we use a serial port, its driver must be open and the port must be configured. **OpenSerial** opens the specified serial port driver (either the modem or the printer port) and kills any pending I/O operations to it. The routine uses the driver's input reference number (-6 for the modem port and -8 for the printer port) to open both the input and output sections, and expects to find the appropriate number on the stack. Figure 1 shows the local stack for the **OpenSerial** routine.

OpenSerial starts by setting up the stack frame and allocat-

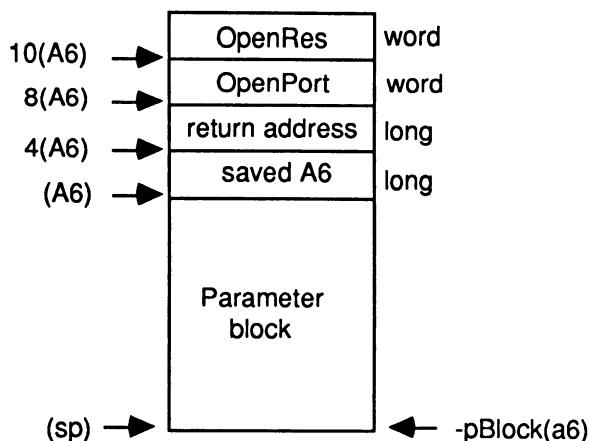


Figure 1 Local stack for **OpenSerial**

ing the space for the parameter block. It then makes register A0 point to the parameter block, as expected by the Device Manager. **OpenSerial** uses the Device Manager **_Open** trap which at most has only three entries in its parameter block. Since **none** of these serial I/O routines are making asynchronous calls to the Device Manager, **OpenSerial** can ignore the **ioCompletion** pointer. It can also ignore the **ioPermsn** byte, leaving only **ioNamePtr**, which is filled after **OpenSerial** checks which port is to be opened. Since both the input and output sides of a port must be opened, **OpenSerial** calls the Device Manager **_Open** trap twice. Once this is done, the routine saves the result code (returned in the parameter block at **ioResult**) on the return stack and calls the Device Manager one more time, to kill any pending I/O operations to the port. Since the parameter block remains unchanged, **OpenSerial** can call the **_KillIO** trap directly. Before returning to the calling program, **OpenSerial** discards the local stack (and the parameter block) and clears the port number from the return stack, leaving only the result word.

The code segment below shows the calling sequence for **OpenSerial** (**PortA** refers to the modem port driver and was set to -6 by an equate statement somewhere else in the program).

```
clr.w    -(sp)      ; room for result
move.w   #PortA, -(sp) ; modem port
jsr      OpenSerial
tst.w    (sp)       ; leave result on stack
bne      Abort
```

Since **OpenSerial** is usually followed by the configuration routine, I leave the result code on the stack (it will be zero if the port opened successfully) and use it to mark the space for the next routine's result code. **Config** resets the serial port and sets it to the new baud rate, data bits, stop bits and parity desired. **Config** doesn't modify some of the other important serial port parameters (mainly handshaking) since the default parameters suited my needs. Like **OpenSerial**, **Config** expects to see the port's input driver reference number on the stack. It also expects to see a configuration word, which represents the baud rate, data bits, stop bits and parity being set. Figure 2 shows the local stack

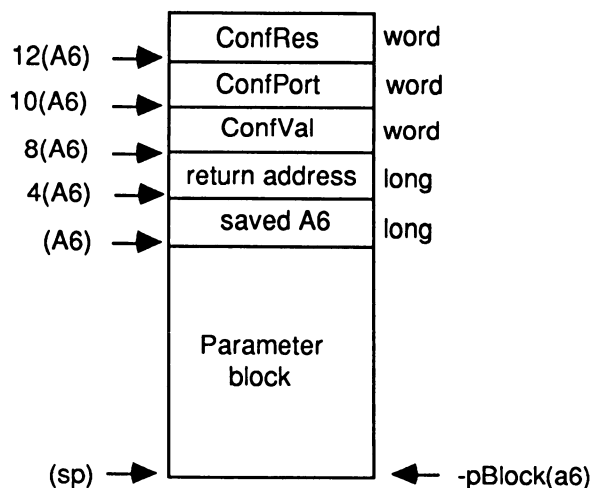


Figure 2 Local stack for **Config**

for Config and the code section below shows its calling sequence.

```

move.w #PortA,-(sp) ; modem port
move.w #Config1200,-(sp)
jsr Config
tst.w (sp) ; leave result on stack
bne Abort

```

Config uses the Device Manager `_Control` trap, which requires a few more parameters than `OpenSerial`. Two parameters, `ioCompletion` and `ioVRefNum` (serial ports don't have volume names) can be ignored, but it must set the `ioRefNum` to that of the port being configured. Config uses the `csCode` parameter to tell the Device Manager to reset the port. The new configuration value is passed in `csParam`.

For some applications, opening and configuring the serial port is all you need to do to initialize it for use. Unfortunately, the default size of input driver's buffer (set by the Device Manager) is a tiny 64 bytes, and this small buffer can become a major problem.

We all know that TextEdit is no speed demon. If you're writing a communications program and use TextEdit to display the incoming text, characters may be received by the port faster than TextEdit can display them. The 64 byte serial input buffer will quickly overflow and you will lose some characters. Luckily, this input buffer can be resized using `SetBuf`. This routine, like `OpenSerial` and `Config`, requires a reference number for the port's input driver. It also requires a long word that represents the buffer's new size.

`SetBuf`, like all the other routines, starts by setting aside some room for the parameter block. It will then request a nonrelocatable block from the Memory Manager to serve as the new input buffer. If `SetBuf` succeeds in getting the new block, it tells the input driver to use this buffer through the Device Manager `_Control` trap. Figure 3 shows the local stack for `SetBuf`.

```

move.w #PortA,-(sp) ; modem port

```

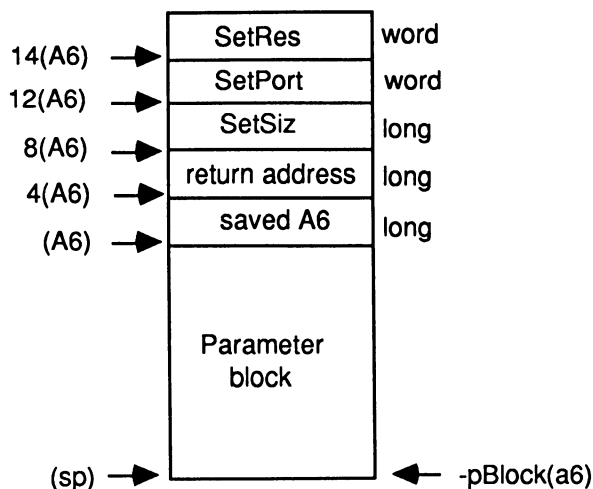


Figure 3 Local stack for `SetBuf`

```

move.l #512,-(sp) ; 512 byte buffer
jsr SetBuf
move.w (sp)+,d0 ; take result off stack
bne Abort

```

In this example, the input buffer is made 512 bytes long. Since `SetBuf` is usually the last step in initializing a serial port, the result word that was floating on the stack from the previous two routines (`OpenSerial` and `Config`) finally gets popped off. As we've come to expect, if `SetBuf` has a problem allocating the memory for the buffer, this word will be non-zero.

It's important to set the input buffer back to the default value before terminating your program (the programmer's motto should be "always leave things the way you found them"). This can be done with `SetBuf` by passing a value of 0 as the size of the buffer. The Device Manager will then set the input buffer back to its default value.

Once the serial port has been opened and initialized and the size of the serial input buffer has been increased, your program is ready to send and receive data through the port.

The input routine `GetSerial` takes two parameters, the port number and a pointer to the buffer that will hold the incoming data (note that this isn't the same as the input driver buffer). `GetSerial` first checks to see if there are any characters waiting to be read by calling `SerGetBuf`, which is a special version of the Device Manager `_Status` trap. This Device Manager call returns a long word containing the number of characters in the driver's input buffer. If the input buffer is empty, `GetSerial` will leave this zero count on the stack and end. If there are characters available (and if you use TextEdit, there will be quite a few backed up), `GetSerial` reads them into the new buffer using the Device Manager `_Read` trap. `GetSerial` then leaves the character count on the stack and ends. Figure 4 shows the local stack for `GetSerial`. Here's an example from the terminal program:

```

clr.l -(sp) ; room for char count
move.w #PortA,-(sp) ; use the modem port
pea 10Buf(a5) ; put data here
bsr GetSerial

```

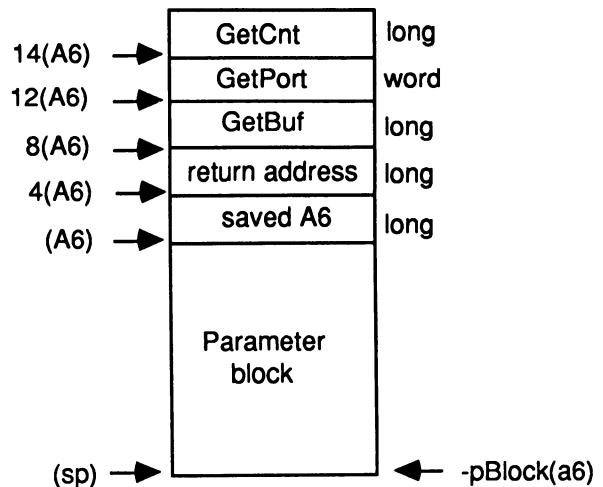


Figure 4 Local stack for `GetSerial`

```
move.l (sp)+,CharCnt(a5) ; save count
```

PutSerial simply shoves data out the port, letting the Device Manager worry about errors (like overflowing the output buffer). Since we are calling PutSerial synchronously, this is not a serious shortcoming and PutSerial will work well, without problems.

PutSerial takes three parameters, the port number, a pointer to the buffer holding the data to be sent and the number of characters to be sent. Even though PutSerial ignores errors, it returns a “no errors” result value to the calling program; this is to keep it consistent with the other routines and to make room for a real error value if a future version of the routine does check for errors. The calling sequence is pretty simple:

```
clr.w    #0,-(sp)    ; room for dummy
move.w   #PortA,-(sp) ; modem port
pea      IOBuf(a5)   ; send data in buffer
pea      CharCnt(a5) ; this many
bsr      PutSerial
move.w   (sp)+,d0    ; pop dummy
```

Figure 5 shows the local stack for PutSerial.

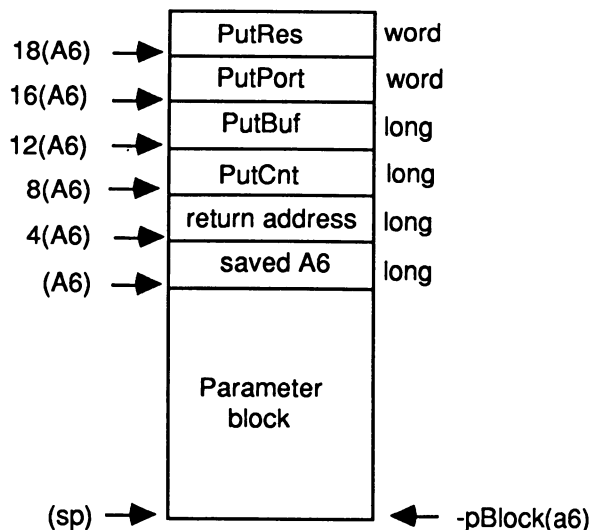


Figure 5 Local stack for PutSerial

These serial I/O routines are lacking in a few features (mainly better and more complete error checking) but since the interface to the calling program (be it a high level language or assembly) is consistent, these routines can be used as the basis for more complete routines, or used as-is, without any major changes to the calling program.

Using the Serial I/O Routines in the Real World: An Example

The easiest way to learn how to use the serial I/O routines is with an example. Term is an *extremely* simple terminal program; it sends characters typed at the keyboard out the modem port and uses TextEdit to display the incoming characters. Besides the standard File and Edit menus, Term has a Command menu with an Erase Screen item and four (300, 1200, 2400 and 9600) baud

settings. It also uses the clipboard, so you can cut and paste text to and from its window. While the text that is pasted to the terminal window is displayed, it is not sent out the port (as my old calculus book would say “the solution to this problem is left to the reader”). The main window in Term is moveable, but not resizable. This is a minor annoyance, which can be easily fixed.

Term will run on almost any Macintosh (it doesn’t run on Macs with the old 64K ROMs) and is structured to serve as the skeleton of a more complex communications program. To maximize speed and reduce the need for global variable storage, most assembly language programmers store frequently used handles and pointers in unused CPU registers. Since I don’t know what you plan to do with Term, I’ve tried to minimize the use of registers for handle storage. Most of the menu and window handles are kept as global variables.

Macintosh applications often share the same basic structure and even some of the initialization and event parsing routines. Term is no exception. It is a quilt of routines that have been borrowed from other applications. Term begins by initializing the modem port and going through the usual ritual of setting up things and initializing managers. It finishes the initialization by copying the contents of the clipboard to the TextEdit scrap. Term ends when a non-zero value is passed to the event loop, either when the user selects Quit from the menu or clicks in the program window’s close box. Term then cleans up, resets the input buffer back to its default size and returns to the Finder.

Since Macintosh programs spend most of their time in the event loop, this is the best place for Term to check the modem port and display any new characters that may have been received. You may notice that the EventRecord is declared and used as a global *variable*. It used to be a common practice to declare the EventRecord as a constant; this saved a few keystrokes when typing in the program. The practice did not cause problems when the program was running because the 68000 doesn’t make any distinctions between the memory used for variable storage and that used for constants and code. With the 68020, 68030 and future CPUs, memory management becomes part of the computer hardware. With an MMU, an operating system could set aside memory to be used as storage for a program’s code and constants... and declare this memory to be read only. Imagine the surprise of a program trying to write to a write-protected EventRecord!

Term Routines

Instead of a detailed description of every routine in Term, I’ll focus on the routines that make it different.

PutScreen filters the characters in the input buffer, IOBuf, and transfers the displayable characters to Outbuf, the output buffer. The variable CharCnt is used to tell TextEdit how many characters are in the output buffer.

A communications program *must* be able to display characters on the screen quickly, especially at the higher baud rates, when data may be flooding in through the serial port. On the Macintosh, the quickest way to get a character on the screen is to use QuickDraw’s DrawChar function, and this is what I used in the early versions of Term. Unfortunately, DrawChar does little

else besides drawing a character on the screen. All text manipulations (simple things like backspace and carriage return) must be handled with a stupendous amount of code. I knew that TextEdit could handle some of these simple functions, and TextEdit had the extra bonus of making it easy to cut and paste the text to the clipboard and back. Unfortunately, TextEdit is extremely slow. My initial attempts to use TextEdit were a miserable failure; I was losing most of the incoming data. The solution, of course, was to increase the size of the serial input buffer.

While TextEdit is an improvement over DrawChar, it will only recognize a limited number of control characters, so PutScreen must deal with any control characters it finds in IOBuf. In this version of PutScreen, only the carriage return (CR), Bell, Tab and backspace (BS) characters are recognized; all other control characters are discarded. By replacing the simple compare and branch code with a lookup table, PutScreen could be made to emulate a real terminal, like the VT-52 or VT-100.

PutScreen lets TextEdit handle carriage returns. To backspace, PutScreen decrements the output buffer pointer by one (to erase the previous character) and decrements the character count by two (to account for the deleted character and the backspace character). If the character count is zero, then no characters have to be displayed, and PutScreen passes control to the NextEvent routine. If CharCnt is greater than zero, there are still characters to be processed, and PutScreen goes on to the next character in IOBuf. If the CharCnt is less than zero, we are no longer dealing with "fresh" characters and must delete a character that has already been displayed. To do this, PutScreen passes the BS character directly to the _TEKey trap, which deletes the last character displayed. It then passes control to the NextEvent loop.

If the control character is a Bell (control G) PutScreen will generate a short beep, decrement the character count, and continue with the filtering.

Tabs are hard coded for five spaces. PutScreen places five spaces in the output buffer and increments the character count by four (we don't want the tab character itself to be displayed) before continuing with the filtering.

Displayable characters are transferred from the IOBuf to the OutBuf, and the filtering continues until all the characters have been dealt with. PutScreen then uses _TEInsert to pass the OutBuf to TextEdit, which displays it. PutScreen also calls _TESelView, to scroll the window, if needed.

KeyDown takes characters typed at the keyboard and saves them in IOBuf. If the command key is pressed, it converts the character to a control character, and inserts it into the IOBuf. KeyDown then sends the character out the modem port.

InBaud is used to change the baud rate. This routine unchecks the previous menu item, checks the new one and sets the new baud rate, using the Config routine.

Possible Enhancements

Like most programs, Term has room for improvements. The easiest additions would be to make the window resizable and make the program MultiFinder friendly.

Term will operate at 19,200 baud; I was just too lazy to add it as a menu option. A menu option to change the default port

might be useful, but dangerous. Since the printer port is used with networks and printers, the routine to switch ports **must** check to see if AppleTalk is active or if the printer port is being used by another program.

A slightly more difficult task would be to have Term send text pasted in it's window, from the Clipboard, out the serial port. Another TextEdit related problem is a cosmetic one; I implemented a screen erase function in Term by clearing out the all the data in the TextEdit record. It works, but it makes the screen flash like an IBM PC screen...somewhat disconcerting.

A final improvement would be to add a file transferring capability, using the MacBinary extension to the XMODEM protocol. This wouldn't be as difficult as it sounds; both the MacBinary and XMODEM protocols are well described. It would be a good idea to increase the input serial buffer to at least 1280 bytes, room enough to fit a 1K XMODEM block. This would speed up the data rates by allowing the program to work on one block of data while another block is being received in the background.

Borrowed Routines

A few of the routines and techniques that I used in Term were borrowed from other programs. The code to deal with the clipboard is a slightly modified version of the code in the Clipboard chapter in Dan Weston's "The Complete Book of Macintosh Assembly Language Programming, Volume II." Some of the ideas on how to handle TextEdit came from Dan's "The Complete Book of Macintosh Assembly Language Programming, Volume I." It should be pretty obvious that these are two very useful (if somewhat dated) books on Macintosh assembly language programming.

Victor Barger's Rose Curve Generator program is an excellent example of good code layout and presentation. I've tried to make Term as easy to read.

References

Dan Weston, "The Complete Book of Macintosh Assembly Language Programming" Volumes I and II.

Victor Barger, "Rose Curve Generator," MacTutor, January 1987

Apple Computer, "Inside Macintosh" volumes I - V (mainly volumes I and II)

Apple Computer, Technical Note 130, ioCompletion

Kirk Austin, "A MIDI Library for Pascal," MacTutor, July 1987

Kirk Austin, "MIDI Demo uses MIDI Library," MacTutor, December 1987

Listing: term.job

```
asm term.asm execedit
link term.link execedit
rmaker term.r termedit
```

Listing: term.link

```
/Output term.code
```

```
/Type 'TEMP' '????'
```

```
term
SerI/O
```

```
;All done
/End
```

Listing: SerI/O.asm

```
; File: SerI/O.asm
```

```
; SerI/O.asm consists of following stand-alone serial I/O
routines:
; Function OpenSerial(Port: integer): OSErr
; Function Config(Port, SerVal: integer): OSErr
; Function GetSerial(Port: integer; buffer: pointer): CharCnt
; Function PutSerial(Port: integer; CharBuf: pointer; count:
longint): OSErr
; Function SetBuf(Port: integer; BufSiz: long): OSErr
; Port refers to port reference number, which is -6 for modem
; input port (port A) or -8 for printer input port (port B)
; For description of how to write modular assembly language
; routines see Weston's "Complete Book of Macintosh Assembly
; Language Programming Volume I", Appendix D
; 1/14/88 Working version of the routines
; written by Frank Henriquez
```

```
; Register usage:
; d0: general purpose.
; d1, d2, d3, d4, d5, d6, d7 : not used.
; a0: general purpose, used for buffers and parameter blocks.
; a1: general purpose.
; a2, a3, a4: not used.
; a6: used as the local stack frame pointer
; a5, a7 : system use.
```

```
Include Traps.D
Include SysEqu.D
```

```
XDEF OpenSerial
XDEF Config
XDEF GetSerial
XDEF PutSerial
XDEF SetBuf
```

```
; equates for each stack frame
```

```
pBlock equ -ioQelSize
```

```
OpenPar equ 2 ; OpenSerial - # of parameter words on stack
OpenRes equ 10 ; OpenSerial - offset to result word
OpenPort equ 8 ; OpenSerial - offset to Port Reference #
```

```
ConfPar equ 4 ; Config - # of parameter words on stack
ConfRes equ 12 ; Config - offset to result word
ConfPort equ 10 ; Config - offset to port Reference #
ConfVal equ 8 ; Config - offset to port config value
```

```
GetPar equ 6 ; you've probably figured the pattern
GetCnt equ 14 ; offset to character count
GetPort equ 12 ; out by now...
GetBuf equ 8 ; offset to character buffer pointer
```

```
PutPar equ 10
PutPort equ 16
PutBuf equ 12 ; offset to character buffer pointer
PutCnt equ 8 ; offset to character count pointer
PutRes equ 18 ; dummy result
```

```
SetPar equ 6
SetRes equ 14 ; offset to SetBuf result word
SetPort equ 12
SetSiz equ 8 ; offset to new buffer size
```

```
; misc. equates
```

```
SerReset equ 8 ; from Inside Macintosh
```

```
SerSetBuf equ 9 ; from Inside Macintosh
ioNamePtr equ 18 ; missing from MDS
```

```
;----- OpenSerial -----
; Opens the modem port for reads and writes.
; Function OpenSerial(Port: integer): OSErr
```

```
OpenSerial
link a6, #pBlock ; local space for parameter block
lea pBlock(a6), a0 ; point to parameter block
lea 'AIn', a1 ; assume port A input
cmpi.w #-6, OpenPort(a6) ; open port B if not -6 (port A)
beq.s @1
lea 'Bin', a1 ; open port B input
@1 bsr.s Openit
lea 'AOut', a1 ; assume port A output
cmpi.w #-6, OpenPort(a6)
beq.s @2
lea 'Bout', a1 ; open port B output
@2 bsr.s Openit
```

```
OExit move.w d0, OpenRes(a6) ; procedure returns a 0 if no
errors
```

```
_KillIO ; kill any pending calls
unlk a6 ; discard local stack area
move.l (sp)+, a0 ; get return address
addq.l #OpenPar, sp ; clean up stack
jmp (a0) ; end of InitSerial
```

```
Openit
move.l a1, ioNamePtr(a0) ; tell the Device Manager which
_Open ; port to open
rts
```

```
;----- Configure ports -----
; Sets the Port to the baud rate, data length,
; parity, stops, etc. held in SerVal.
; Function Config(Port, SerVal: integer): OSErr
```

```
Config
link a6, #pBlock ; local space for parameter block
lea pBlock(a6), a0 ; point to parameter block
move.w ConfPort(a6), d0 ; get Input Reference number
bsr.s doConf ; configure the input side
tst.w d0 ; check for errors and
bne.s CExit ; exit with the error flag in d0
move.w ConfPort(a6), d0 ; get Input Ref number
subq.w #1, d0 ; and make it the output ref num
bsr.s doConf ; configure the output side
```

```
CExit move.w d0, ConfRes(a6) ; save result
unlk a6
move.l (sp)+, a0
addq.l #ConfPar, sp ; and clean up the stack
jmp (a0) ; end of Configure
```

```
doConf move.w d0, ioRefNum(a0) ; port #
move.w #SerReset, csCode(a0) ; reset port to
move.w ConfVal(a6), csParam(a0) ; these new settings
_Control
rts
```

```
;----- SetBuf -----
; increases the size of the Port input buffer
; Function SetBuf(Port: Integer; BufSiz: long): OSErr
```

```
SetBuf link a6, #pBlock ; local space for parameter block
move.l SetSiz(a6), d0
beq.s @1 ; if size = 0, reset input buffer
_NewPtr ; get a nonrelocatable block
tst.l d0 ; abort if couldn't get it
bne Exit
move.l a0, a1 ; save block ptr in a1
move.l SetSiz(a6), d0 ; reload the buffer size
```

```

@1lea pBlock(a6),a0 ; point to parameter block
move.w SetPort(a6),ioRefNum(a0)
move.w #SerSetBuf,csCode(a0)
move.l a1,csParam(a0) ; pointer to new buffer
move.w d0,csParam+4(a0)
_Control
Exit move.w d0,SetRes(a6)
unlk a6 ; standard exit routine
move.l (sp)+,a0
addq.l #SetPar,sp
jmp (a0) ; end of SetBuf

;----- GetSerial -----
; GetSerial checks the Port to see if any
; characters have been received, and proceeds
; to read them into the input buffer.
; If none have been received, GetSerial returns
; a 0 in the character count variable.
; Function GetSerial(Port:integer; buffer:pointer) : CharCnt:
long

GetSerial
link a6,#pBlock
lea pBlock(a6),a0
move.w GetPort(a6),ioRefNum(a0)
move.w #2,csCode(a0) ; call SerGetBuf
_Status
move.l csParam(a0),d0 ; get # of characters received
move.l d0,GetCnt(a6)
beq.s @2 ; if none, then leave, if there are
; characters available, read them.
@1move.w GetPort(a6),ioRefNum(a0)
move.l GetBuf(a6),ioBuffer(a0) ; the input buffer
move.l d0,ioReqCount(a0) ; read the characters
_Read
@2unlk a6 ; standard exit routine
move.l (sp)+,a0
addq.l #GetPar,sp
jmp (a0) ; end of GetSerial

```

```

;----- PutSerial -----
; PutSerial sends the characters in the buffer
; out the Port. This routine is pretty
; primitive - it ignores any pending writes
; and errors, but...it works. It returns a dummy result.
; function PutSerial(Port:integer; CharBuf:pointer; count:
; pointer): OSErr

PutSerial
link a6,#pBlock ; local space for parameter block
lea pBlock(a6),a0 ; point to parameter block
move.w PutPort(a6),d0 ; get the port ref #
subq.w #1,d0 ; make it the port output ref num
move.w d0,ioRefNum(a0) ; modem output port
move.l PutBuf(a6),ioBuffer(a0) ; point to buffer
move.l PutCnt(a6),a1 ; get pointer to char count
move.l (a1),ioReqCount(a0) ; and put char count here
_Write
move.w #0,PutRes(a6) ; save a 0 as a dummy result
unlk a6 ; standard exit routine
move.l (sp)+,a0
adda.l #PutPar,sp
jmp (a0) ; end of PutSerial

```

Listing: term.asm

```

; File: Term.asm
;
; Term.asm is simple terminal program, shows how to use low
; level serial I/O routines.
; written by Frank Henriquez
; code that handles transfers between clipboard and Text Edit
; is based on the code in chapter 3 of Dan Weston's "Complete
; Book of Macintosh Assembly Language Programming Vol. II"

```

```

; Some coding ideas were borrowed from the program "Rose.Asm"
; by Victor Barger, appeared in Jan. 1987 issue of MacTutor.
; 7/15/87 Initial Turbo Pascal version, using Quickdraw.
; 9/1/87 assembly language translation.
; 10/10/87 modified to use TextEdit.
; 10/18/87 fully functional version.
; 10/20/87 text handling improved, Baud rates added.
; 1/14/88 modified to use external serial I/O routines.

```

```

; Register usage:
; d0: general purpose, also holds the menu#.
; d1: general purpose, also menu item# and the Modify value.
; d2: general purpose.
; d3, d4, d5, d6, d7 : not used.
; a0: general purpose, used for buffers and parameter blocks.
; a1: general purpose.
; a2: holds TextEdit handle.
; a3: general purpose.
; a4, a6 : not used.
; a5, a7 : system use.

```

```

; pre-compiled routines from Serial I/O.asm

```

```

XDEF OpenSerial
XDEF Config
XDEF GetSerial
XDEF PutSerial
XDEF SetBuf

```

```

; Standard include files

```

```

Include Traps.D
Include Toolsequ.D
Include Sysequ.D
Include Quicsequ.D

```

```

;----- Program constants -----

```

```

AppleMenu equ 1 ; Apple
AboutItem equ 1 ; First item in Apple menu

```

```

FileMenu equ 2 ; File
QuitItem equ 1 ; First item in File menu

```

```

EditMenu equ 3 ; Edit
UndoItem equ 1 ; Items in Edit menu
CutItem equ 3
CopyItem equ 4
PasteItem equ 5
ClearItem equ 6

```

```

CmdMenu equ 4 ; Command menu
EraseItem equ 1 ; Erase screen

```

```

B300 equ 3 ; 300 baud
B1200 equ 4 ; 1200 baud
B2400 equ 5 ; 2400 baud
B9600 equ 6

```

```

AboutDialog equ 1 ; About dialog
ButtonItem equ 1 ; First item in DITL

```

```

WindID equ 1

```

```

PortA equ -6
PortB equ -8

```

```

baud300 equ 380
baud1200 equ 94
baud2400 equ 46
baud9600 equ 10
stop10 equ 16384
noParity equ 8192
data8 equ 3072

```

```

Config300    equ baud300 + data8 + stop10 + noParity
Config1200   equ baud1200 + data8 + stop10 + noParity
Config2400   equ baud2400 + data8 + stop10 + noParity
Config9600   equ baud9600 + data8 + stop10 + noParity

```

;----- Program starts here -----

```

Start
  bsr.s InitSerial
  bsr InitManagers
  bsr SetupMenu
  bsr SetupWindow
  bsr SetupTextEdit
  bsr CliptoTE

```

```

EventLoop
  _SystemTask      ; check for DA calls
  move.l a2,-(sp)   ; get handle to text record
  _TEIdle          ; blink cursor etc.
  bsr GetSerCh      ; get a character from the modem port
  bsr PutScreen     ; and put it on the screen
  clr.w -(sp)
  move.w #$0fff,-(sp) ; look for all events
  pea EventRecord(a5)
  _GetNextEvent     ; get the next event (WaitNextEvent!?)
  move.w (sp)+,d0
  beq.s EventLoop  ; loop until an event
  bsr HandleEvent   ; take care of the event
  beq.s EventLoop  ; time to quit if not zero
  move.l a2,-(sp)
  _TEDispose
  clr.w -(sp)
  move.w #PortA,-(sp) ; modem port
  clr.l -(sp)        ; reset default serial buffer
  jsr SetBuf
  addq.l #2,sp
  _ExittoShell

```

;----- InitSerial -----
; Open serial port for reads and writes, set
; the baud rate to 1200 baud and increase the
; size of the serial input buffer to 512 bytes

```

InitSerial
  clr.w -(sp)      ; room for routine results
  move.w #PortA,-(sp) ; open the modem port (Port A)
  jsr OpenSerial
  tst.w (sp)        ; if Open failed, beep and quit.
  bne.s Abort
  move.w #PortA,-(sp)
  move.w #Config1200,-(sp) ; set modem port to 1200 baud
  jsr Config
  tst.w (sp)        ; if Config failed, beep and quit.
  bne.s Abort
  move.w #PortA,-(sp) ; make the serial input buffer
  move.l #512,-(sp) ; 512 bytes big. Prevents data loss.
  jsr SetBuf
  move.w (sp)+,d0    ; clean up the stack, and abort if
  bne.s Abort        ; SetBuf failed.
  rts

```

;----- Abort -----
Abort
 move.w #30,-(sp)
 _Sysbeep
 _ExittoShell ; jump back to Finder

;----- InitManagers -----
InitManagers
 _MoreMasters ; prevents heap fragmentation
 move.l #\$ffffff,d0
 _NewHandle ; compact heap
 pea -4(a5) ; QuickDraw globals
 _InitGraf
 _InitFonts

```

  move.l #$0000ffff,d0 ; flush all events
  _FlushEvents
  _InitWindows
  _InitMenus
  clr.l -(sp)          ; no restart procedure
  _InitDialogs
  _TEInit
  _InitCursor
  rts

```

;----- SetupMenu -----
SetupMenu
;Apple menu:
 clr.l -(sp)
 move.w #AppleMenu,-(sp)
 _GetRMenu ; get Apple menu from the rsrc file
 move.l (sp),AppleMHD1(a5) ; save handle to menu
 move.l (sp),-(sp)
 clr -(sp)
 _InsertMenu
 move.l #'DRVR',-(sp)
 _AddResMenu ; add DA's to Apple menu

;File menu:
 clr.l -(sp)
 move.w #FileMenu,-(sp)
 _GetRMenu
 move.l (sp),FileMHD1(a5) ; save handle to menu
 clr -(sp)
 _InsertMenu

;Edit menu:
 clr.l -(sp)
 move.w #EditMenu,-(sp)
 _GetRMenu
 move.l (sp),EditMHD1(a5) ; save handle to menu
 clr -(sp)
 _InsertMenu

;Command menu:
 clr.l -(sp)
 move.w #CmdMenu,-(sp)
 _GetRMenu
 move.l (sp),CmdMHD1(a5) ; save handle to menu
 clr -(sp)
 _InsertMenu
 move.l CmdMHD1(a5),-(sp) ; put handle on stack
 move.w #B1200,d0
 move.w d0,BaudChk(a5)
 move.w d0,-(sp)
 move.w #-1,-(sp) ; next to the 1200 baud item
 _CheckItem
 _DrawMenuBar
 rts

;----- SetupWindow -----
SetupWindow
 clr.l -(sp)
 move.w #WindID,-(sp)
 pea WindowStorage(a5)
 move.l #-1,-(sp) ; make it the top window
 _GetNewWindow ; load it from the rsrc file
 move.l (sp),WindowHdl(a5) ; save handle
 _SetPort ; make it the current port
 move.w #monaco,-(sp) ; Monaco
 _TextFont
 move.w #9,-(sp) ; in 9 pt.
 _TextSize
 rts

;----- SetupTextEdit -----
SetupTextEdit
 clr.l -(sp) ; space for text handle
 pea DestRect ; DestRect Rectangle

```

pea ViewRect      ; ViewRect Rectangle
_TENew
move.l (sp)+,a0    ; save text handle
move.l a0,a2      ; in a2
move.l (a0),a0
move.b #-1,teCROnly(a0)
move.w #-1,-(sp)
move.l a2,-(sp)   ; enable TE auto scroll (128K ROM)
_TEAutoView
rts

;----- CliptoTE -----
; CliptoTE copies text in the clipboard to the TE scrap

CliptoTE
move.l #0,a3      ; check for TEXT in the desk scrap
bsr.s #1
bmi.s #2          ; leave if none

move.l #0,d0      ; lets get the scrap, but first
_NewHandle        ; get a handle for it
move.l a0,a3
bsr.s #1
move.l TEScrpHandle,a0 ; replace old handle to TEScrap
_DisposHandle
move.l a3,TEScrpHandle ; with ours
move.l TEScrpHandle,a0
_GetHandleSize     ; get length of our scrap
move.w d0,TEScrpLength
@2rts              ; leave CliptoTE

@1clr.l -(sp)      ; space for result
move.l a3,-(sp)
move.l #'TEXT',-(sp) ; just check for TEXT
pea GSOffset(a5)
_GetScrap
move.l (sp)+,d0    ; return result
rts

;----- GetSerCh -----
GetSerCh
clr.l -(sp)        ; room for character count
move.w #PortA,-(sp) ; if modem port has data, save it
pea IOBuf(a5)      ; in this buffer.
jsr GetSerial
move.l (sp)+,CharCnt(a5) ; get # of characters received.
bra NextEvent

;----- PutScreen -----
; PutScreen will display the characters in
; the buffer, taking care of some control
; characters. It uses TextEdit for display.
PutScreen
move.l CharCnt(a5),d1 ; skip if no characters in buffer
beq NextEvent
subq.l #1,d1          ; adjust character count for loop
PutTE lea IOBuf(a5),a0 ; point to the character buffers
lea Outbuf(a5),a1
TELoop move.b (a0)+,d0 ; get a char from the input buffer
andi.w #$7f,d0        ; clear out 8th bit
cmpi.w #$20,d0        ; if less than space, then it
blt.s CtrlCh          ; must be a Ctrl character
inBuf move.b d0,(a1)+ ; printable character
inLoop dbrad1,TELoop
pea Outbuf(a5)         ; print the sanitized output buffer
move.l CharCnt(a5),-(sp)
move.l a2,-(sp)
_TEInsert
move.l a2,-(sp)
_TESelView            ; scroll if needed
bra NextEvent         ; exit PutScreen

;--- handle control characters ---
; Control characters are serviced here. A
; lookup table would be more efficient, in
; particular if we were to emulate a real

```

```

; terminal (like a VT-52 or VT-100).

CtrlCh cmpi.w #$0d,d0 ; let Text Edit take care of CR
beq.s inBuf
cmpi.w #8,d0          ; backspace
beq.s BS
cmpi.w #7,d0          ; bell
beq.s Bell
cmpi.w #9,d0          ; tab
beq.s Tab
subq.l #1,CharCnt(a5) ; ignore anything else
bra.s inLoop

BSsubq.l #1,a1        ; move back outbuf pointer by 1
subq.l #2,CharCnt(a5) ; 1 for BS char, 1 for erased char
beq NextEvent         ; no chars to print - all done.
bgt.s inLoop          ; continue filtering

move.w d0,-(sp)       ; use TEKey to force backspace
move.l a2,-(sp)       ; if backed out of input buffer
_TEKey
bra NextEvent

Bell move.w #30,-(sp) ; make a beep
_SysBeep
subq.l #1,CharCnt(a5) ; get rid of character
bra.s inLoop

Tab move.w #4,d2       ; 5 spaces for a tab
@1move.b #$20,(a1)+   ; put them into the output buffer
dbrad2,@1
addq.l #4,CharCnt(a5) ; adjust the character count
bra.s inLoop          ; ( # spaces, -1 for the tab char)

;----- Handle Event -----
; This routine is the core of the program.
; The event number is used as an index into
; the EventTable. These entries cover all
; the events that could happen while the
; program is in the main loop.

HandleEvent
move.w Modify(a5),d1
move.w What(a5),d0    ; get event number
add.w d0,d0           ; *2 for table index
move.w EventTable(d0),d0 ; point to routine
jmp EventTable(d0)    ; and jump to it

EventTable
dc.w NextEvent-EventTable ; 0 (Null)
dc.w MouseDown-EventTable ; 1 mouse down
dc.w NextEvent-EventTable ; 2 mouse up (Not used)
dc.w KeyDown-EventTable   ; 3 key down
dc.w NextEvent-EventTable ; 4 key up (Not used)
dc.w KeyDown-EventTable   ; 5 auto-key
dc.w Update-EventTable    ; 6 update
dc.w NextEvent-EventTable ; 7 disk inserted (Not used)
dc.w Activate-EventTable  ; 8 activate
dc.w NextEvent-EventTable ; 9 abort (Not used)
dc.w NextEvent-EventTable ; 10 network (Not used)
dc.w NextEvent-EventTable ; 11 I/O driver (Not used)
dc.w NextEvent-EventTable ; 12 app1Evt (Not used)
dc.w NextEvent-EventTable ; 13 app2Evt (Not used)
dc.w NextEvent-EventTable ; 14 app3Evt (Not used)
dc.w NextEvent-EventTable ; 15 app4Evt (suspend/resume?)

;----- Mouse down -----
MouseDown
clr.w -(sp)           ; space for result
move.l Point(a5),-(sp) ; get mouse coordinates
pea WWindow(a5)       ; Event Window
_FindWindow
move (sp)+,d0         ; get region number

```



```

add d0,d0 ; *2 for index into table
move.w WindowTable(d0),d0 ; point to routine offset
jmp WindowTable(d0) ; jump to routine

WindowTable
dc.w NextEvent-WindowTable ; In Desk (Not used)
dc.w InMenu-WindowTable ; In Menu Bar
dc.w SystemEvent-WindowTable ; In System Window
dc.w Content-WindowTable ; In Content
dc.w Drag-WindowTable ; In Drag
dc.w NextEvent-WindowTable ; In Grow (Not used)
dc.w QuitRoutine-WindowTable ; In Go Away
dc.w NextEvent-WindowTable ; In Zoom in box (Not used)
dc.w NextEvent-WindowTable ; In Zoom Out box (Not used)

;----- in Menu -----
InMenu
clr.l -(sp) ; make room on stack
move.l Point(a5),-(sp) ; mouse event
_MenuSelect
move.w (sp)+,d0 ; save menu
move.w (sp)+,d1 ; and menu item

Choices ; called by command key too
cmp.w #AppleMenu,d0
beq.s InAppleMenu
cmp.w #FileMenu,d0
beq.s InFileMenu
cmp.w #EditMenu,d0
beq.s InEditMenu
cmp.w #CmdMenu,d0
beq InCmdMenu

ChoiceReturn
bsr UnHiliteMenu ; unhighlight the menu bar
bra NextEvent

;----- in InAppleMenu -----
; In the Apple menu. If it wasn't About, it must have been a
; desk accessory. If so, open the desk accessory.

InAppleMenu
cmp.w #AboutItem,d1 ; is it About?
beq.s About ; if so go do About...
move.l AppleMHD1(a5),-(sp) ; look in Apple Menu
move.w d1,-(sp)
pea DeskName(a5) ; get Item Name
_GetItem
clr -(sp) ; space for opening result
pea DeskName(a5) ; open Desk Acc
_OpenDeskAcc
move.w (sp)+,d0 ; pop result

GoSetOurPort
bsr SetOurPort
bra.s ChoiceReturn

;----- About -----
; Set up the About dialog box, and wait for
; the proper click or keypress. End by
; closing the dialog box and setting the
; port to us.

About
clr.l -(sp) ; space for dialog pointer
move #AboutDialog,-(sp) ; dialog rsrc #
pea DStorage(a5)
move.l #-1,-(sp) ; dialog goes on top
_GetNewDialog
move.l (sp),-(sp) ; copy handle for Close
_SetPort ; make dialog box the port
move.l a2,-(sp)
_TEDeActivate

```

```

WaitOK
clr.l -(sp) ; clear space for handle
pea ItemHit(a5)
_ModalDialog ; wait for a response
move.w ItemHit(a5),d0 ; look to see what was hit
cmp.w #ButtonItem,d0 ; was it OK?
bne.s WaitOK
_CloseDialog ; handle already on stack
bra.s GoSetOurPort

;----- in FileMenu -----
inFileMenu
cmp.w #QuitItem,d1 ; is it Quit?
bne.s ChoiceReturn ; no, go get next event
bsr UnHiliteMenu ; unhighlight the menu bar
bra RealQuit ; go Quit

;----- in EditMenu -----
; The Edit Menu support routines transfer the
; TE scrap to the clipboard, and back. See the
; clipboard chapter in Dan Weston's
; "Assembly Language Programming Volume II"

InEditMenu
bsr SystemEdit ; Desk accessory active?
bne.s ChoiceReturn ; yes, SystemEdit handled it
cmp.w #CutItem,d1
beq.s Cut
cmp.w #CopyItem,d1
beq.s Copy
cmp.w #PasteItem,d1
beq.s Paste
cmp.w #ClearItem,d1
beq.s ClearIt
bra ChoiceReturn

Cut
move.l a2,-(sp)
_TECut ; Cut and copy text
bra.s Convert

Copy
move.l a2,-(sp)
_TECopy ; Copy text to scap
bra.s Convert

Paste
bsr CliptoTE ; from the clipboard
move.l a2,-(sp)
_TEPaste ; Paste
bra ChoiceReturn

ClearIt
move.l a2,-(sp)
_TEDelete ; Clear without copying

Convert
bsr.s TEtoClip ; copy to clipboard
bra ChoiceReturn

;----- TEtoClip -----
; TEtoClip copies the TE scrap to the Clipboard

TEtoClip
move.l TEScrpHandle,a0
_GetHandleSize ; get length of our scrap
move.l d0,a3 ; save length
clr.l -(sp) ; space for result
_ZeroScrap ; want to write over scrap
move.l (sp)+,d0
move.l TEScrpHandle,a0
_Hlock ; lock pointer to our scrap
clr.l -(sp) ; space for result

```

```

move.l a3,-(sp)
move.l #'TEXT',-(sp) ; just check for TEXT
move.l TEScrpHandle,a0
move.l (a0),-(sp) ; save pointer to TE scrap
_PutScrap
move.l (sp)+,d0
move.l TEScrpHandle,a0
_HUnlock ; unlock handle
rts ; leave TEtoClip

;----- in CmdMenu -----
InCmdMenu
cmp.w #EraseItem,d1 ; baud rate change?
bne.s InBaud

; it wasn't baud change command, fall through to erase screen

move.l #0,-(sp) ; set Selection range to cover
move.l #ffff,-(sp) ; everything in this TE block.
move.l a2,-(sp)
_TESetSelect
bra.s ClearIt ; and clear them out

;----- in BaudMenu -----
InBaud
cmp.w #B300,d1
beq.s @1
cmp.w #B1200,d1
beq.s @2
cmp.w #B2400,d1
beq.s @3
cmp.w #B9600,d1
beq.s @4
bra NextEvent

@1move.w #Config300,SerCon(a5)
bra.s @5
@2move.w #Config1200,SerCon(a5)
bra.s @5
@3move.w #Config2400,SerCon(a5)
bra.s @5
@4move.w #Config9600,SerCon(a5)
@5move.l CmdMHdl(a5),-(sp)
move.w BaudChk(a5),-(sp) ; uncheck previous item
move.w d1,BaudChk(a5) ; save new baud item
move.w #0,-(sp)
_CheckItem
move.l CmdMHdl(a5),-(sp) ; put handle on stack
move.w BaudChk(a5),-(sp) ; and place a check
move.w #-1,-(sp) ; next to the current baud
_CheckItem
clr.w -(sp)
move.w #PortA,-(sp) ; set the new baud rate
move.w SerCon(a5),-(sp)
jsr Config
move.w (sp)+,d0 ; ignore the result
bra ChoiceReturn

;----- UnhilitMenu -----
UnhilitMenu
clr.w -(sp) ; all menus
_HiLiteMenu
rts

;----- SystemEdit -----
SystemEdit
clr -(sp) ; space for result
move.w d1,-(sp) ; get item in Edit menu
subq #1,(sp) ; SystemEdit is off by 1
_SysEdit
move.b (sp)+,d0
rts

;----- SystemEvent -----

```

```

SystemEvent
pea EventRecord(a5)
move.l WWindow(a5),-(sp)
_SystemClick ; let the system do it
bra NextEvent

;----- in Content -----
Content
clr.l -(sp) ; room for result
_FrontWindow
move.l (sp)+,d0 ; front window pointer
cmp.l WindowHdl(a5),d0 ; same as our pointer?
beq.s @1 ; ignore it
move.l WWindow(a5),-(sp)
_SelectWindow ; do it, fall through to NextEvent
bra NextEvent

@1pea Point(a5)
_GlobalToLocal
move.l Point(a5),-(sp)
btst #9,d1
sne d0
move.b d0,-(sp)
move.l a2,-(sp)
_TEClick
bra NextEvent

;----- in Drag -----
Drag
move.l WWindow(a5),-(sp)
move.l Point(a5),-(sp)
pea WBounds
_DragWindow ; drag window and fall through
bra NextEvent

;----- Quit Routine -----
QuitRoutine
clr.w -(sp) ; room for result
move.l WindowHdl(a5),-(sp) ; get pointer to open window
move.l Point(a5),-(sp) ; mouse point
_TrackGoAway
move.w (sp)+,d0
beq NextEvent ; didn't really want to quit

RealQuit
move.l WindowHdl(a5),-(sp) ; did want to quit
_CloseWindow
move.w #-1,d0 ; d0 is TRUE - ok to quit
rts

;----- Key down -----
KeyDown
move.w Message+2(a5),d0 ; get key code and character
btst #8,d1 ; command key pressed?
beq.s SaveCh ; yes, take care of it.
andi.w #$001f,d0 ; convert to a control character
SaveCh move.b d0,I0Buf(a5) ; save it in the buffer
move.l #1,CharCnt(a5) ; send the character typed
clr.w -(sp) ; room for dummy result
move.w #PortA,-(sp) ; in the input buffer
pea I0Buf(a5)
pea CharCnt(a5)
jsr PutSerial ; and send it out the modem port
move.w (sp)+,d0 ; get rid of dummy result
bra.s NextEvent

;----- Update -----
Update
move.l WindowHdl(a5),-(sp)
_BeginUpdate
pea ViewRect ; erase visible window
_EraseRect
pea ViewRect
move.l a2,-(sp)

```

```

_TEUpdate
move.l WindowHdl(a5),-(sp)
_EndUpdate
bra.s NextEvent

;----- Activate -----
Activate
move.l WindowHdl(a5),d0
cmp.l Message(a5),d0 ; our window?
bne.s NextEvent
btst #0,d1 ; activate?
beq.s Deactivate ; no

move.l a2,-(sp) ; move Text Handle To Stack
_TEActivate
move.l EditMhdl(a5),-(sp) ; get handle to the menu
move.w #UndoItem,-(sp) ; disable 1st item (undo)
_DisableItem

SetOurPort
move.l WindowHdl(a5),-(sp)
_SetPort
bra.s NextEvent

; Deactivate window, turn off TextEdit, enable undo for the
desk accessories.

Deactivate
move.l a2,-(sp) ; get Text Handle
_TeDeactivate
move.l EditMhdl(a5),-(sp) ; get handle to the menu
move.w #UndoItem,-(sp) ; enable 1st item (undo)
_EnableItem
; bra NextEvent ; *CAUTION!* if more code
; is added after this routine, remember to uncomment the line
above!

;----- Next Event -----
NextEvent
moveq #0,d0 ; say that it's not Quit
rts ; return to EventLoop

;----- Window stuff -----
ViewRect dc.w 5,4,290,500; Text Record's View Rect
DestRect dc.w 5,4,290,500; Text Record's Dest Rect
WBounds dc.w 5,5,335,510

;----- Event variables -----
; IMPORTANT! remember, the EventRecord is
; very much a variable and NOT a constant.
; Never declare the EventRecord variables
; with a dc command.

EventRecord ds.w 0
What: ds.w 1 ; event number
Message: ds.l 1
When: ds.l 1
Point: ds.l 1 ; mouse coordinates
Modify: ds.w 1 ; key and button state

;----- Application variables -----
AppleMhdl ds.l 1 ; handle for apple menu
FileMhdl ds.l 1 ; handle for file menu
EditMhdl ds.l 1 ; handle for edit menu
CmdMhdl ds.l 1 ; handle for command menu

DStorage ds.w DWindLen
DeskName ds.w 16 ; DA name
ItemHit ds.w 1
WWindow ds.l 1
WindowStorage ds.w WindowSize
WindowHdl ds.l 1 ; handle to the window
GSOffset ds.l 1 ; Get Scrap Offset dummy

```

```

;Serial port variables
IOBuf ds.b 512 ; general I/O buffer
OutBuf ds.b 512 ; sanitized version of IOBuf
SerCon ds.w 1 ; current serial port config
BaudChk ds.w 1 ; current checked item
CharCnt ds.l 1 ; # of characters read

```

end

Listing: term.r

* This is the resource file for the program "Term"
* the \CA character generates a "hard" space.
* Give it a type APPL and creator FHMT.

Term
APPLFHMT

* Get the code portion.

INCLUDE term.code

TYPE MENU

,1
About Term...
(-

,2
File
Quit

,3
Edit
Undo
(-
Cut
Copy
Paste
Clear

,4
Command
Erase screen
(-
1200 Baud
2400 Baud
9600 Baud

Type DLOG
,1

100 100 200 350
Visible NoGoAway
1
0
1

Type DITL
,1
2

Button
60 100 90 150
OK

StaticText
15 40 59 238
Simple Terminal Program\0D\CAFrank Henriquez 1/88

TYPE WIND
,1
Terminal
44 7 337 507
Visible GoAway
0
0



Assembly Lab

Real-Time Driver



Jeff E. Mandel, MD MS
New Orleans, LA
Volume 5, Number 10

A Real-Time Device Driver in MPW Asm

When working in the laboratory or in engineering, it is often necessary to use the Mac to control serial devices. Such devices are generally configured to respond to simple commands, and provide a simple response. An example would be the American Edwards AccuPro™ Volumetric Infusion Pump, which is a device for delivering intravenous infusions of drugs. This device contains an RS-232 interface, supports full duplex 2400 baud communication, and has a simple command language for setting and interrogating the device's infusion rate, the limits for infused volume and infusion time, the volume already infused, and the status of the pump.

Programming the pump involves four steps; the command string is built in memory, a pointer to this string is placed in a parameter block for a `_PBWrite`, a `_PBRead` is used to obtain the pump's response, and the returned string is decoded. While this may seem simple, one must remember that the serial communication takes a significant amount of time, and could take forever if the pump or the serial line fails. Additionally, the serial communications controller is capable of functioning with minimal CPU supervision, which transpires at interrupt level, and thus, it would be nice to use asynchronous device driver calls, to free the Mac to work on keeping the user interface going, while the serial controller handles the IO in the background.

The program I will describe for performing these tasks is a driver. Most Mac programmers are familiar with drivers from writing DAs, however, a device driver is somewhat of a different animal, particularly when one wishes it to be "real-time" without intervention from the foreground program. The device driver has several purposes:

- 1) To insulate the application from the peculiarities of the device, so that the device can be changed without requiring a change in the application. Additionally, the device could be simulated by a driver, so that debugging of application code can be done without having to have the physical device.

- 2) To insulate the application programmer from having to understand how serial communications, real-time IO, etc. work.

- 3) To permit all of the resources associated with the IO task to be grouped together, so that they can be installed and uninstalled easily.

In writing such a driver, it is important to recall several important factors:

- 1) Drivers cannot utilize application globals for their own storage.

- 2) Drivers are only notified of events that pertain to them when they own the frontmost window, and then only receive mouse, keyboard, update, and activate events.

- 3) Activities which occur at interrupt level (i.e. `ioCom-`

pletion routines, Vertical retrace tasks, Time manager tasks, etc.) are extremely limited in what they can do, due to the uncertainty of the heap.

In order to write such a driver, several decisions were made:

- 1) The driver utilizes `app1Evs` to signal that it needed to regain control. Thus, the serial reads and writes utilize an `ioCompletion` routine which simply posts an `app1Evt`.

- 2) Serial driver calls have a time limit. When the serial call is queued (asynchronously), a Time manager task is primed. If the serial IO completed first, it stops the Time manager task from completing by setting the `tmCount` field to zero, and if the read or write did not complete within the specified time, the time manager task issues a `_KillIO` call on the relevant serial driver. This results in the `ioCompletion` routine being called, which notifies the driver. The time manager task is contained in a resource of type 'TMMR', which is loaded into the heap, locked, dereferenced, and the pointer placed in the `tmAddr` field of the time manager queue entry. The resource also contains a long-word of private storage, which holds a pointer to a private parameter block (used for the `KillIO` call).

- 3) `App1Evs` are posted with the parameter block pointer of the completed serial call as the message. The pump driver places the parameter block pointer of the pending pump driver in the `ioMisc` field of the serial call parameter block so the downstream routines can keep track of it. Alternatively, the downstream routines could get this pointer from the `dCuQHead` field of the DCE.

- 4) `App1Evs` are passed to the driver by a `GetNextEvent` filter. This routine is called at the end of the `_GetNextEvent` trap, with `A1` pointing to the `EventRecord`, and boolean result in both `D0` and at `4(A7)` (See Macintosh Technical Note #85). If the what field of the `EventRecord` equals `app1Evt`, the routine examines the message, and if it contains a negative word in the `ioRefNum` field of the parameter block pointed to by the message, it processes the event. If the event is processed, the filter sets the boolean result to False, so the the application knows not to deal with it. In either case, the routine exits by a `JMP` to the previous contents of the `JGNEFilter` global. This pointer is stored locally in our `GNEFilter` proc.

- 5) The `GNEFilter` is a separate resource (type = 'GNEF'), which is loaded in the pump driver Open routine. The resource is patterned after the `DRVR` resource, in that it contains an offset to the code as its first word, and some local storage is provided between this word and the beginning of the code. The `GNEFilter` is loaded into the heap during the driver Open routine, using a `_GetNamedResource` call. This is done so that several drivers can share the `GNEFilter`, since only the first driver will load the resource into the heap. The `GNEFilter` keeps track of which

resources have opened it by keeping their reference numbers in the local storage. Duplicate entries are avoided. The GNEFilter also keeps a parameter block pointer in its private storage for its private use.

6) When the GNEFilter processes an `app1Evt` with a negative `ioRefNum`, it accesses the parameter block pointer in the `ioMisc` field of the parameter block referenced in the message field of the event, and checks to see that the `ioRefNum` of this PB is in the list of cooperating drivers in the GNEFilter private storage. If it is, then we place the event pointer in the `csParam` field of the GNEFilter's private parameter block, set the `csCode` to `accEvent`, and the `ioRefNum` to reference number of the `ioMisc` parameter block (this will be the `ioRefNum` of our driver; the `ioRefNum` of the parameter block passed in the message field will be one of the serial drivers). We then issue an immediate `_Control` call.

7) The `Control` routine supports four `csCodes` – `accEvent`, `accRun`, `KillIO`, and `GoodBye`. `Goodbye` simply `JMPs` to the `Close` routine. `KillIO` issues `KillIO` calls on the serial drivers and returns. `AccRun` asynchronously queues a `Status` call on the driver to inquire the pump status (thus, irrespective of what the application does, it will be notified of the pump status periodically). The `accEvent` `csCode` is generated by our GNEFilter, and is used to allow the driver to respond to serial IO completion at event level (that is, when the heap is consistent). The routine examines the low nibble of the `ioTrap` field of the parameter block pointed to by the event message. If the trap was `_Write`, the routine sets up the parameter block for single character reads on the serial input channel and queues an asynchronous `_Read`. If the trap was `_Read`, the routine examines the character read. If the character is a carriage return (ASCII 13), the accumulated input buffer is sent to be digested, if it is an asterisk (*), it is ignored (the McGaw pump generates these periodically to let us know it is pumping), and if it is any other character, it is appended to the input buffer. In the latter two cases, the routine queues another asynchronous `_Read`. In the first case, the routine posts an `app2Evt` with the message the pointer to the original parameter block used to queue the status call, and exits via `JIODone`.

8) The `Status` routine uses the three words in the `csParam` field of the parameter block to figure out what the application wants the driver to send to the pump. The `Request` field is used to look up a single character in a table (which is loaded from a resource of type 'PDDF'). This character specifies which of the pump functions is to be interrogated or set, as specified in the `Action` field. The `Info` field contains the numeric value to be passed to the pump and/or returned to the application. The pump in general wants decimal integers, but in some cases wants a hex word, and the formatting information is specified in the table. Having constructed the string to be sent to the pump, the routine allocates a parameter block, places the string pointer in the `ioBuffer` field, the pointer to the driver `Status` parameter block in the `ioMisc` field, fills in the `ioCompletion` address, and queues an asynchronous `_Write`. Note that if the `noQueueBit` of the trap is set, the `Status` routine places the request at the head of the queue. It does this by checking the queue, and if there are zero or one queue entries, issuing the `_Status` call asynchronously, but if

there are two or more queue entries, it slips the request after the current queue head by changing the `qLink` fields of the `qHead` PB and the PB in question.

9) The `Open` routine loads the GNEFilter and time manager task (as detailed above), opens the serial drivers and configures them for the pump, loads the pump request table, allocates the `dCtlStorage` handle, and sets up the driver globals there.

10) The `Close` routine removes the time manager task, closes the serial ports, removes the driver's reference number from the GNEFilter private storage, and restores the previous GNEFilter if it is the last value there. It then deallocates all the pointers and handles it allocated and exits.

The application thus needs only do the following things to utilize the driver:

- 1) Open the driver
- 2) Allocate a pointer for the parameter block, setting the three words in the `csParam` field to pass the desired function, and queue the `_Control` call. The call should be made asynchronously. Immediate calls can be used to "jump the line", that is, make sure the call is the next one to be taken from the queue.
- 3) The event loop should handle `app2Evts` by first processing the information in the `ioResult` and `csParam` fields, then disposing of the pointer.
- 4) If the pump driver is to function irrespective of what the application is doing, the application must frequently call `_GetNextEvent` with the event mask `app1Evt` mask set. In order to guarantee this you will need a `FilterProc` for `ModalDialogs` and `Alerts` which calls `_GetNextEvent` with the event mask set to `app1Mask` when the routine receives a null event, and a `DragHook` and `MenuHook` which calls `_GetNextEvent` with the event mask set to `app1Mask`. Additionally, any compute-intensive routines might include a call to the `DragHook` routine. The alternative is to keep track of driver calls and repost the ones that time out.
- 5) Call `_SystemTask` if you want periodic events.
- 6) Prior to closing the driver, you should do one of two things:
 - a) Issue a `_KillIO` call on the driver to flush the queue.
 - b) Wait until the queue empties itself. This can be done by waiting until the `dCtlQHead` field of the driver's DCE is zero.

This is necessary because the `_Close` trap sits and waits for the driver to complete the pending request. The driver cannot complete the request, however, without the event loop, so we hang forever.

7) While the driver is set up to handle goodbye kisses to close the structures, it is much safer to close the driver explicitly, due to the serious adverse consequences of exiting without restoring the `JGNEFilter` pointer. The truly paranoid can load the GNEFilter into the system heap. The same caveat applies to the time manager queue entry (the pointer, not the routine) – if this is deallocated but not removed from the time manager queue, bad stuff will happen. I have not extensively worked on making the driver "safe as milk", in general, during debugging, if it crashed, I rebooted. But then, I just got my Mac II.

8) The program is written in MPW assembler, and uses the structured programming macros. I have hacked up some of these

to make them work properly for writing drivers. The only significant change is in DRVRExit, which checks the noQueueBit of the argument, and if it is clear, exits via the JIODone vector, otherwise, RTS.

The code for the driver, the GNEFilter, and the timer task, as well as the rez files and the shell commands to build the driver are presented below. Note that I actually build the driver as a desk accessory. This is done so that I can install it with the DA Mover. This is easier during development, but once the driver is debugged, it can be changed with the resource editor.

The sources have been hacked for the sake of brevity. All INCLUDES were deleted, many equates omitted, and code specific to the pump deleted as well. If you really want to assemble this, get the source disk.

```
TITLE 'Pump driver'
ParamBlockSize equ 100
ActionOffset equ csParam
RequestOffset equ csParam+2
InfoOffset equ csParam+4
```

```
EnabledFlags equ (1<<dStatEnable) \
+ (1<<dCtlEnable) + (1<<dNeedTime)
```

```
EventMask equ 1<<app1Evt
```

```
DStore Record 0
WriteUnit DS.W 1
ReadUnit DS.W 1
TimeCount DS.L 1
KillBlock DS.L 1
KillTask DS.L 1
IncomingLength DS.B 1
IncomingString DS.B 9
Align 2
TableOffset equ *
TableLen DS.W 1
Table DS.W 5
ENDR
```

```
IOString Record 0
OutgoingString DS.B 8
ReadBuffer DS.B 1
Align 2
StringAllocation equ *
ENDR
```

```
DRVStart DRVEntry
```

```
DRVBegin Save=A2-A4/D1-D2,\
with=(DStore,IOString,GNEGlobals);
```

```
DC.B EnabledFlags
DC.B 0
DC.W 7*60 ; 7 seconds
DC.W EventMask
DC.W 0 ; No menu
```

```
DC.W DRVROpen
DC.W DRVPrime
DC.W DRVRCtrl
DC.W DRVStatus
DC.W DRVRClose
```

```
DRVTitle
DC.B 'PumpDriver'
ALIGN 2
```

```
DRVROpen DRVEnter
MOVE.L A1,A2
```

```
* Load the pump request table from the
* resource fork
```

```
Call _GetNamedResource:L ('PDDF':L, #'Table':A
),A4:L
EXG A0,A4
_HLock
_GetHandleSize
If# D0 LT.L #0Then.S
MOVE.W #openErr,ioResult(A4)
Return
EndIf#
EXG A0,A4
MOVE.L D0,D1
MOVE.L A0,-(SP)
```

```
* Get a new handle to put the Driver
* globals and the pump request table into
```

```
MOVE.L A1,-(SP)
ADD.L #TableOffset,D0
_NewHandle ,clear
_HLock
MOVE.L A0,dCtlStorage(A2)
MOVE.L (A0),A1
MOVE.L A1,A3
ADD.L #TableOffset,A1
MOVE.L (ResHandle),A0
MOVE.L D1,D0
_BlockMove
Call _ReleaseResource ( ResHandle:L)
MOVE.L (SP)+,A1
```

```
* Open the serial drivers and set them up
* for the pump (I have omitted the .AIN code
* since it is identical)
```

```
MOVE.L #ParamBlockSize,D0
_NewPtr ,clear
LEA #'AOUT',A2
MOVE.L A2,ioNamePtr(A0)
MOVE.B #fsWrPerm,ioPermsn(A0)
_Open
If# D0 NE.W #noErr Then.S
MOVE.L (SP)+,A0
MOVE.W #openErr,ioResult(A0)
Return
EndIf#
MOVE.W ioRefNum(A0),
WriteUnit(A3)
```

```
* Set the output port for 2400 baud, 1
* stop, no parity, eight bit communication
```

```
MOVE.W #PortSetting,csParam(A0)
MOVE.W #8,csCode(A0)
_Control
```

```
_DisposPtr
```

```
* Set up a parameter block for the serial
* IO timeout function. The pointer is
* stored in the Driver private storage.
* Insert task into the time manager queue
* which performs the IO timeout function.
```

```
MOVE.L #ParamBlockSize,D0
_NewPtr ,clear
MOVE.L A0,KillBlock(A3)
MOVE.L #1500,TimeCount(A3)
MOVE.L #tmQSize,D0
_NewPtr
MOVE.L A0,KillTask(A3)
Call TimeOutInstall ( A0:L ,
```

```

KillBlock(A3):L )

* Install the GNEFilter

Call _GetNamedResource:L ( #'GNEF':L
, #'GNEFilter':A ),A2

CMPA #0,A2
If# NE Then.S
MOVE.L (A2),A2
MOVE.W Drvr_num(A2),D1
If# D1 EQ.W #0 Then.S
MOVE.W (A2),D0
LEA (A2,D0.W),A3
MOVE.L JGNEFilter,-4(A3)
MOVE.L A3,JGNEFilter
MOVE.L #ParamBlockSize,D0
_NewPtr ,clear
MOVE.L A0,Control_Ptr(A2)
EndIf#
MOVE.W dCtlRefNum(A1),D2
If# D1 LE #entry_slots Then.S
For# D1 DownTo #1 Do.S
If# Drvr_num(A2,D1.W*2) EQ.W
D2 Then.S
GoTo#.S DuplicateDrvr
EndIf#
EndF#
ADD.W #1,Drvr_num(A2)
MOVE.W Drvr_num(A2),D1
MOVE.W D2,
Drvr_num(A2,D1.W*2)
Else#.S

* Get the Resource ID for the driver to
* calculate the ID of owned Alert
* (sub ID 0)

LEA DRVEntry,A0
_RecoverHandle
LINK A6,#-12
Call _GetResInfo ( A0:L , (A6):A ,
4(A6):A , 8(A6):L )
MOVE.W (A6),D2
UNLK A6
MULU #32,D2
ADD.W #0wned,D2
Call _CautionAlert:W ( D2:W ,
0:L ),CC
EndIf#
EndIf#
DuplicateDrvr:
MOVE.L (SP)+,A0
MOVE.W #openErr,ioResult(A0)
Return

DRVRClose DRVEnter
MOVE.L A0,-(SP)
MOVE.L dCtlStorage(A1),A2
MOVE.L (A2),A3

MOVE.L #ParamBlockSize,D0
_NewPtr ,clear
MOVE.W ReadUnit(A3),ioRefNum(A0)
_Close
MOVE.W WriteUnit(A3),
ioRefNum(A0)
_Close
_DisposPtr

MOVE.L KillTask(A3),A0
_RmvTime
_DisposPtr
MOVE.L KillBlock(A3),A0
_DisposPtr

```

```

Move.L A2,A0
_DisposHandle

MOVE.W dCtlRefNum(A1),D2
Call _GetNamedResource:L (
#'GNEF':L , #'GNEFilter':A ),A2
CMPA #0,A2
If# NE Then.S
MOVE.L (A2),A3
MOVE.W Drvr_num(A3),D1
If# D1 GT.W #1 Then.S
While# D2 NE.W
Drvr_num(A3,D1.W*2) Do.S
DBEQ.W D1,DuplicateClose
EndW#
If# D1 NE.W Drvr_num(A3) Then.S
ADD.W #1,D1
For# D1To Drvr_num(A3)
Do.S
LEA Drvr_num(A3,D1.W*2),
A4
MOVE.W (A4),-2(A4)
EndF#
EndIf#
SUBQ.W #1,Drvr_num(A3)
ElseIf#.S D1 EQ.W #1 Then.S
If# D2 NE.W Drvrentries(A3)
Then.S
GoTo#.S DuplicateClose
EndIf#
MOVE.L GNE_Next(A3),
JGNEFilter
Call _ReleaseResource ( A2:L )
EndIf#
EndIf#
DuplicateClose:
MOVE.W #noErr,D0
MOVE.L (SP)+,A0
DRVRExit ioTrap(A0)

DRVRPrime DRVEnter
MOVE.W noErr,D0
Return

DRVRStatus DRVEnter

* First, check to see if this is an
* immediate call. If so, it "jumps the
* line", and will be the next call serviced
* after the current one (if any) is
* completed.

MOVE.W ioTrap(A0),D1
BTST #noQueueBit,D1
If# NE Then.S ;Immediate call
LEA dCtlQHead(A1),A2
CMP.L #0,(A2)
If# EQ Then.S ;queue empty
_Status ,async
Return
EndIf#

MOVE.L (A2),D1
CMP.L 4(A2),D1
If# EQ Then.S ; only one entry
_Status ,async
Return
EndIf#

* Fool the driver into thinking this is an
* asynchronous queue entry

MOVE.W ioTrap(A0),D1

```

```

BCLR      #noQueueBit,D1
BSET      #asyncTrpBit,D1
MOVE.W    D1,ioTrap(A0)
MOVE.W    #1,ioResult(A0)

* Get parameter block of queue head, put
* qLink of qHead in current parameter
* block;and replace it with current
* parameter block pointer

MOVE.L    (A2),A2
MOVE.L    (A2),(A0)
MOVE.L    A0,(A2)
Return
EndIf#

MOVE.L    A0,A2
MOVE.L    dCtlStorage(A1),A4
MOVE.L    (A4),A4

MOVE.W    TableLen(A4),D1
MOVE.W    RequestOffset(A2),D0
If#      D0 GT.W D1 Then.S ;Not a valid
                                ;pump call

MOVE.W    #statusErr,ioResult(A2)
MOVE.L    A2,D0
MOVE.W    #app2Evt,A0
_PostEvent
MOVE.L    A2,A0
MOVE.L    #statusErr,D0
DRVRExit  ioTrap(A0)
EndIf#

MOVE.L    #ParamBlockSize,D0
_NewPtr ,clear

* The next section of code places a pointer
* to the needed string, and the number of
* characters, into the parameter block. It
* was omitted from this listing.

MOVE.W    WriteUnit(A4),
          ioRefNum(A0)
LEA      SerialComplete,A3
MOVE.L    A3,ioCompletion(A0)
_Write ,async

* Set up the time manager task to KillIO
* on the channel in TimeCount
* milliseconds. KillBlock is a previously
* allocated parameter block.

MOVE.L    KillBlock(A4),A0
MOVE.L    WriteUnit(A4),
          ioRefNum(A0)
MOVE.L    KillTask(A4),A0
MOVE.L    TimeCount(A4),D0
_PrimeTime
MOVE.L    #noErr,D0
Return

DRVRCtrl  DRVREnter

If#      csCode(A0) EQ.W #accEvent      Then
MOVE.L    csParam(A0),A2
MOVE.L    message(A2),A2
MOVE.W    ioTrap(A2),D2
AND.W     #$00FF,D2
MOVE.W    ioResult(A2),D3

* Check to see the serial call was not
* aborted by KillIO. If so, inform the
* application, dequeue the pump driver
* call, and return

```

```

If# D3      EQ.W #abortErr      Then.S
MOVE.L    ioMisc(A2),A0
EXG      A0,A2
_DisposPtr
MOVE.W    #TimeOut,ioResult(A2)
ADD.W     D2,ioResult(A2)
MOVE.L    A2,D0
MOVE.W    #app2Evt,A0
_PostEvent
MOVE.L    #TimeOut,D0
MOVE.L    A2,A0
DRVRExit  ioTrap(A0)
EndIf#

* The serial call did not time out, so
* figure out if it was a READ or a WRITE
* by examining the low nybble of the trap
* field of the parameter block.

If# D2      EQ.W #aRdCmd      Then

* READ - examine the character read. If it
* is a '*', we ignore it and queue another
* read. If it is a carriage return, we call
* RespondToRead to interpret the
* completed pump response. Any other
* character is incorporated into the string
* being built at IncomingString and we
* increment the IncomingLength byte,
* then queue another READ. If we exceed
* 8 characters, we have a problem.

MOVE.L    ioBuffer(A2),A3
MOVE.B    (A3),D1
MOVE.L    dCtlStorage(A1),A4
MOVE.L    (A4),A4
If#      D1 EQ.B #13      Then.S
MOVE.L    KillTask(A4),A0
CLR.L     tmCount(A0)
MOVE.L    ioMisc(A2),A3
Call RespondToRead
MOVE.L    ioMisc(A2),A0
EXG      A0,A2
_DisposPtr
MOVE.L    A2,D0
MOVE.W    #app2Evt,A0
_PostEvent
CLR.B     IncomingLength(A4)
CLR.L     IncomingString(A4)
CLR.L     IncomingString+4(A4)
CLR.L     D0
MOVE.L    A2,A0
DRVRExit  ioTrap(A0)
ElseIf# S D1 EQ.B #'*'      Then.S
MOVE.L    A2,A0
_Read ,async
Return
Else# S
MOVE.B     IncomingLength(A4),
          D0
EXT.W     D0
MOVE.B     D1,
          IncomingString(A4,D0.W)
ADD.W     #1,D0
If#      D0 LE.W #8      Then.S
MOVE.B     D0,
          IncomingLength(A4)
Else# S
MOVE.L    KillBlock,A0
MOVE.W    ReadUnit(A4),
          ioRefNum(A0)
_KillIO
MOVE.L    ioMisc(A2),A0

```



```

        EXG A0,A2
        _DisposPtr
        MOVE.L A2,D0
        MOVE.W #app2Evt,A0
        _PostEvent
        MOVE.L A2,A0
        MOVE.L
            #TooManyCharacters,D0
        DRVRExit                                ioTrap(A0)
    EndIf#
    MOVE.L A2,A0
    _Read ,async
    Return

    EndIf#
ElseIf#.S D2 EQ.W #aWrCmd Then.S
    MOVE.L dCtlStorage(A1),A4
    MOVE.L (A4),A4
    MOVE.L KillBlock(A4),A0
    MOVE.W ReadUnit(A4),
            ioRefNum(A0)

    MOVE.L A2,A0
    MOVE.W ReadUnit(A4),
            ioRefNum(A0)

    MOVE.L #1,ioReqCount(A0)
    MOVE.L ioBuffer(A0),A2
    LEA ReadBuffer(A2),A2
    MOVE.L A2,ioBuffer(A0)
    LEA SerialComplete,A4
    MOVE.L A4,ioCompletion(A0)
    _Read ,async
    Return
    EndIf#
ElseIf#.S csCode(A0) EQ.W #accRun
    Then.S
    MOVE.L A0,-(SP)
    MOVE.L #ParamBlockSize,D0
    _NewPtr ,clear
    MOVE.W dCtlRefNum(A1),
            ioRefNum(A0)

    MOVE.W #S_request,
            RequestOffset(A0)

    MOVE.W #Ask,ActionOffset(A0)
    CLR.L ioCompletion(A0)
    _Status ,async
    MOVE.L (SP)+,A0
ElseIf#.S csCode(A0) EQ.W #killCode
    Then.S
    MOVE.L #ParamBlockSize,D0
    MOVE.L dCtlStorage(A1),A2
    MOVE.L (A2),A2
    _NewPtr ,clear
    MOVE.W ReadUnit(A2),
            ioRefNum(A0)

    _KillIO
    MOVE.W WriteUnit(A2),
            ioRefNum(A0)

    _KillIO
    _DisposPtr
    Return
ElseIf#.S csCode(A0) EQ.W #goodBye
    Then.S
    JMP DRVRClose
    EndIf#

    MOVE.L #noErr,D0
    DRVRExit                                ioTrap(A0)
    ENDP
    END

```

The following is the GNEFilter Code:

```

    TITLE 'GNE filter'

    StackFrame    RECORD (A6Link),DECR
    Result        DS.W 1

```

```

RetAddr        DS.L 1
A6Link         DS.L 1
LocalSize      equ *
                ENDR

    MAIN
    WITH EventRecord,StackFrame
        ,GNEGlobals

    Entry        DC.W GNEFilter                ;Offset to
                ;GNE filter
                DCB.B GNEGlobalSize,0

```

GNEFilter:

* If the event is not an app1Evt, go to the
 * next event in the GNE filter chain,
 * which we previously stored in GNE_Next

```

    LEA            Entry,A0
    CMP.W #app1Evt,what(A1)
    BNE.S Out

```

* We have an app1Evt. Check to see if the
 * ioRefnum is negative.

```

    LINK A6,#LocalSize
    MOVEM.L A2/D1-D2,-(SP)
    MOVE.L message(A1),A2
    MOVE.L ioMisc(A2),A2
    MOVE.W ioRefNum(A2),D1
    If# GE Then.S
        GoTo#.S PreOut
    EndIf#

```

* We have a negative ioRefnum. Check to
 * see if it is in the list of ioRefnums
 * we are cooperating with. If it is, pass
 * the event to the driver by setting
 * up a parameter block for an immediate
 * call to the control routine.

```

    MOVE.L Control_Ptr(A0),A2
    LEA Drvr_num(A0),A0
    MOVE.W (A0)+,D2

```

Loop:

```

    For# D2 DownTo #1 Do.S
        MOVE.W (A0)+,D0
        If# D0 EQ.W D1Then.S
            MOVE.W D1,ioRefNum(A2)
            MOVE.W #accEvent,csCode(A2)
            MOVE.L A1,csParam(A2)
            MOVE.L A2,A0
            _Control ,immed
            CLR.W D0
            MOVE.W D0,Result(A6)
            Leave#.S Loop
        EndIf#
    EndF#

```

```

    PreOut:
        MOVEM.L (SP)+,A2/D1-D2
        UNLK A6
        LEA            Entry,A0

```

```

    Out:
        MOVE.L GNE_Next(A0),A0
        JMP (A0)
    ENDP
    END

```

The following is the timeout task:

```

    TITLE 'Timeout for serial IO'

    Main

```

KillStore DC.L 0

KillRoutine:

```
LEA      KillStore,A0
MOVE.L (A0),A0
_KillIO
RTS
ENDP
END
```

The following contains the timeout routine installation, and the serial ioCompletion routine

TITLE 'Timeout for serial IO'

```
Export Procedure TimeOutInstall
      ( KillTask:L , KillPtr:L )
Begin      Save=A0-A2
```

```
Call _GetNamedResource:L
      ( #'TMMR':L , #'KillRoutine':A ),A1
CMPA #0,A1
If# NE      Then.S
MOVE.L (A1),A1
MOVE.L KillTask(FP),A0
MOVE.L KillPtr(FP),(A1)
LEA 4(A1),A1
MOVE.L A1,tmAddr(A0)
_InstTime
EndIf#
Return
```

ENDP

* Serial Complete - This routine is the
* ioCompletion routine for the

* asynchronously serial IO calls from the
* pump driver. On completion, the routine
* posts an appIEvt with the message the
* parameter block of the completed serial
* call.

```
Export Procedure SerialComplete
Begin      Save=A5
```

```
MOVE.L CurrentA5,A5
MOVE.L A0,D0
MOVE.W #appIEvt,A0
_PostEvent
Return
ENDP
END
```

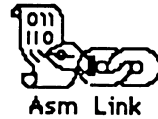
The following are the shell commands to build this as a desk accessory and install it with the DA mover.

```
Asm TimeOut.a
Asm KillRoutine.a
link KillRoutine.a.o 0
-o PumpDriver -ra KillRoutine=16 0
-sn 'Main=KillRoutine' 0
-rt TMMR=-15296
Asm GNEFilter.a
link GNEFilter.a.o 0
-o PumpDriver -sn 'Main=GNEFilter' 0
-ra GNEFilter=16 -rt GNEF=-15296
Asm Pdriver.a
link Pdriver.a.o TimeOut.a.o -t DFIL 0
-c DMOV -o PumpDriver -da 0
-sn 'Main=PumpDriver' -rt DRV=34
"HD 40:System Folder:Font/DA Mover" 0
PumpDriver test_drvr
```



Assembly Lab

Assembly Language for the Rest of Us



Jeffrey B. Kane, MD
Boston, MA
Volume 5, Number 12

Introduction and Purpose

I've always found that it's the little things in life that hang me up. The grandiose projects somehow seem to fall in place, but when I want to do a simple project, using only *obvious* facts, I find that those facts must be *so* obvious, no one has bothered to write them down.

Most people who program do so to create little tools. As an engineer or scientist might we write our code in a high level language like Pascal, C, Fortran, or LISP in order to efficiently read and debug our code (especially with tools from people like Think Technologies and Coral Software). Once the code is up and running, we may notice that one of our procedures, let's say one that crunches a lot of numbers using SANE¹, is taking up most of the processing time. By rewriting this one part of the program we can speed things up dramatically, often 10 times or more!

This article is about assembly language. It's not about writing a lightening fast game or a new version of ExcelTM, rather it's about using the minimum amount of programming we can get away with in order to write a short but useful little bit of code. Neither is it about writing a stand alone program in assembly, but instead we will take a program written in Pascal and rewrite just one of its procedures in assembly language, so the entire program can run much faster.

What we will do

In this article will recode a small subroutine whose mission in life is to take an array and multiply each of its elements by a scalar constant. Although the program is not the most dramatic use of assembly language, and the code we write will not be the most elegant, hopefully it will be clear and easy to understand². We will be illustrating the steps needed to interface any code with a running Pascal program. I chose to make use of the 68881 floating point processor³ in our program, not to add complexity, but to illustrate how easy it is to use this chip. Motorola has made the 68881 appear simply to be an extension of the 68000 CPU⁴. To the programmer the combination of a 68000 and a 68881 appear as a souped up 68000, just like the basic version but turbocharged for numeric speed with a few more instructions and registers added to it. If you don't have this chip the same principles of programming apply. You would simply replace the 68881 instructions with a few more 68000 instructions, but the programming logic would be the same.

MPW Assembler, LSP, and MDS

I will be using LightSpeed Pascal® to develop our main Pascal program. As I explained above the basic program creates an array, then multiplies each number within the array by a

constant. LightSpeed is a great product for writing quick and dirty programs, since you can debug and observe your variables as you go. It even has an assembly level debugger built right in, so if you want to peek at the actual memory to see exactly what your instructions are doing, you can. This is a *great* way to learn the simple and predictable way each line in your program will effect memory.

We will also be using MacsbugTM, a source level debugger that should be available either from your local computer dealer⁵, most user groups, bulletin boards, and/or commercial language products (including LightSpeed). With Macsbug we can look at everything the Mac is doing, while it follows our assembly language instructions as they execute, one line at a time. Tracing our code as it executes will make it clear that each instruction has very predictable and simple results (Remember that if a compressed piece of sand called the 68000 can understand what these instructions mean, so can you!).

Finally we will be using the assembler built into the MPW shell⁶ (version 2.02). Consulair also makes a stand alone Assembler that was the standard of the industry⁷.

In this article we will assume⁸ that you know a what a *bit* is (0 or 1), that 8 bits are stored in a *byte*, that 16 bits make up a *word*, and finally that 32 bits make up a *long word*⁹. We also assume that have some idea of what a hexadecimal number is¹⁰.

The Pascal Code

First lets take a look at the Pascal Code:

```
PROGRAM TestSM;
USES
    SANE;
TYPE
    element = PACKED RECORD
        empty : integer;
        n : extended;
    END;
    vector = ARRAY[0..19] OF element;
    matrix = RECORD
        rows : integer;
        columns : integer;
        vecPtr : ^vector
    END;
VAR
    inVect : vector;
    outVect : vector;
    i : integer;
    inMatrix, outMatrix : matrix;
    scalar : extended;
    error : integer;

FUNCTION ScaleMult (scalar : extended; VAR
    inMat, outMat : matrix) : integer;
EXTERNAL;
```

```

BEGIN
  ShowText;
  scalar := 35;
  inMatrix.rows := 0;
  inMatrix.columns := 19;
  inMatrix.VecPtr := @inVect;
  outMatrix.rows := 0;
  outMatrix.columns := 19;
  outMatrix.VecPtr := @outVect;
  writeln('scalar = ', scalar);
  writeln('scalar, inMatrix, outMatrix, error, inVect');
  write(longint(@scalar), longint(@inMatrix),
  longint(@outMatrix), longint(@error));
  writeln(longint(@inVect));
  FOR i := 0 TO 19 DO
    BEGIN
      inVect[i].empty := 255;
      inVect[i].n := i;
    END;
  error := ScaleMult(scalar, inMatrix, outMatrix);
  writeln('error = ', error);
  writeln('i:5, inVect:10, outVect:10');
  FOR i := 0 TO 19 DO
    writeln(i:5, inVect[i].n:10, outVect[i].n:10);
  writeln('scalar = ', scalar);
  writeln('The End...');
END.

```

In this article, we will rewrite the function ScaleMult in assembly language. From our perspective, all that Pascal does is hand our assembly language program the information in the form we declared in the interface statement. Since we have declared our subroutine as a *function*, our responsibility will be to return a result to Pascal, also in the form we declared. Pascal does not care how we perform our task, only that we get it done.

The first unusual item that you may notice is our definition of an *element*. In most programming, numbers with decimal places in them are defined as “real” numbers. An example of a real number would be 3.14159 or “3.14159 E+0”. The last part, “E+0” means that we will multiply our number “3.14159” times ten raised to the the zero power (which is equal to one). We could have simply written the number as 3.14159 and Pascal would have interpreted it the same way. If the number contained “E+1” then we would have multiplied our base number by ten raised to the first power (or 10), or we could have written the equivalent real number as 31.4159. Similarly “E+3” would have meant multiplying by 10³ (or 1000) and been equivalent to 3,141.59. SANE uses a more accurate form of the real number, called an *extended* number. The extended format can hold more decimal places than a real number, and it can also raise a number to a larger power than a real number (a real number can run between ±3.4E+38 and ±1.2E-38, whereas extended numbers can (approximately) extend from ±1.1E+4932 and ±1.7E-4932, with more decimal places stored for each number). All of those extra decimal places translate into more accuracy as we do our calculations.

In our program we have defined an element to be:

```

element = PACKED RECORD
  empty : integer;
  n : extended;

```

```

END;

```

Instead of just holding an extended number, our element has an integer (which is empty) placed in front of it. We have done this to provide for the difference between the way SANE and the 68881 handle extended numbers. Although we will explain the reasoning below (it's actually not hard to understand), for now simply consider each *Element* of the array as an *extended* Format number.

Basic Assembly Language

Now lets take a look at our assembly language function:

```

MC68881 ; We will be using
; instructions for
; the 68881 chip
MACHINE MC68020 ; optimize for the
; 68020, if you don't
; have one remove
; this line

ScaleMult FUNC EXPORT
; this lets the rest
; of the world know
; about our function

_Debugger OPWORD $A9FF ; this will pop us
; into Macs bugs so
; we can follow every
; instruction and
; observe the
; registers

; The following are all displacements so we
; can address our data on the stack, relative
; to the address stored in A6

result EQU 20 ; two bytes for our
; integer result
Scalar EQU 16 ; 4 bytes (address of
; the scalar)
inMat EQU 12 ; 4 bytes (address of
; the input Matrix)
outMat EQU 8
; 4 bytes (address of
; the output Matrix)
ReturnAdd EQU 4 ; 4 bytes
superScalar EQU -12
; 12 bytes (hold a
; place for our 68881
; version of the
; scalar)
oldA2 EQU -14 ; old the old value
; of A2
oldA3 EQU -16 ; old the old value
; of A3

; *** Add the following features (to make
; this a "real" program):
; 1) make sure the number of
; rows and columns in the input ; Matrix jives with
that of the ; the output matrix.
; 2) Fix the error message to
; tell the Pascal Program if
; there was a problem, such as
; if: rows X columns ≠ vector
; size

; ***** The first "real" line of our program ;
(it's about time!)

```

```

_Debugger ; jump into Macsbugs
; so we can follow
; what's going on

; move all the initial parameters off the
; stack from the calling routine "function
; ScaleMult (scalar:extended;var inMatrix,
; outMatrix:matrix):error"

link A6,#-12 ; push A6 (the frame
; pointer) onto the
; stack, load SP
; into A6, then
; subtract off 12
; bytes from A7 to
; make room for our
; variables

; *****
; after the link instruction
; the stack looks as follows:
;
; high
; integer (2 bytes)
; ----- <- 20(A6)
; scalar (4 bytes)
; ----- <- 16(A6)
; inMatrix (4 bytes)
; ----- <- 12(A6)
; oldMatrix (4 bytes)
; ----- <- 8(A6)
; returnAddress (4 bytes)
; ----- <- 4(A6)
; old A6 (4 bytes)
; ----- <- (A6)
; superscalar (12 bytes)
; ----- <- -12(A6)
; and initial (A7)
; low
;
; *****

; now move the addresses onto the chip so we
; can work with them

MOVE.L A2,-(A7); push A2 onto the
; stack to save it
MOVE.L A3,-(A7); do the same to A3
MOVE.L inMat(A6),A2
; get the pointer to
; the address of
; inMatrix
MOVE.L 4(A2),A0 ; get the address of
; the input matrix
; (4 bytes from the
; start of our matrix
; data structure )
MOVE.L outMat(A6),A2
MOVE.L 4(A2),A1; get the address of
; the output matrix
MOVE.L inMat(A6),A2 ; get the number of
; rows (first byte of
; Matrix record)
MOVE.W (A2),D0
MOVE.W 2(A2),D1; then get the number
; of columns (3rd
; byte of matrix
; record)

MOVE.L scalar(A6),A2
; we will now move
; the scalar to a
; place with a bit

```

```

; more room
LEA superScalar(A6),A3
CLR.W (A3)+
MOVE.L (A2)+,(A3)+ ; move the first 4
; bytes

MOVE.L (A2)+,(A3)+; move the next 4
; bytes

MOVE.W (A2),(A3); move the last byte

LEA superScalar(A6),A2
MOVE.L (A2),D2 ; get the top
; long-word of
; the scalar, shift
; it, then put it
; back
LSL.L #8,D2
LSL.L #8,D2
MOVE.L D2,(A2)

FMOVE.X (A2),FP1; get the modified
; scalar and store it
; on the 68881

Loop MOVE.L (A0),D2 ; get the current
; high Long word of
; the element from
; the input matrix

LSL.L #8,D2
LSL.L #8,D2
MOVE.L D2,(A0)

FMOVE.X (A0),FP0; get the current
; element
FMUL.X FP1,FP0 ; multiply by the
; scalar
FMOVE.X FP0,(A1) ; put the element
; back in RAM
MOVE.L (A1),D2 ; now shift the high
; Long word back (in
; the output
; matrix, and then put
; it away in RAM
LSR.L #8,D2
LSR.L #8,D2
MOVE.L D2,(A1)
MOVE.L (A0),D2 ; now shift the high
; Long word back (in
; the input matrix)
; and then put it
; away
LSR.L #8,D2
LSR.L #8,D2
MOVE.L D2,(A0)

ADD #12,A0 ; increment the
; element's address
; (to get the next
; element)
ADD #12,A1

DBLT D1,Loop ; decrement the
; number of columns
; and test
; if we are here we have gone through one
; column
MOVE.L inMat(A6),A2
MOVE.W 2(A2),D1; restore the number
; of columns
DBLT D0,Loop ; test if we have
; completed through
; all the rows
; if we are here, we have gone through all

```

```

; the rows

; since we have done our work let's put away
; our toys (clear the stack of all the
; garbage) and go home.

MOVE.L (A7)+,A3      ; pop off the old A3
MOVE.L (A7)+,A2      ; pop off the old A2
UNLK A6
MOVE.L (A7)+,A0      ; save the return
                     ; address
ADD.L #12,A7         ; move the stack
                     ; pointer clear all
                     ; the data off the
                     ; stack

CLR.W (A7)           ; replace the return
                     ; value on the stack
                     ; with ours

; ** NOTE: we are pushing a value of zero
; onto the stack, meaning that nothing went
; wrong. This is bogus and in the real
; version we need to fix this!

MOVE.L A0,-(A7); push the return
                     ; address back onto
                     ; the stack

RTS                  ; return to reality
                     ; our calling
                     ; program)

ENDF
END

```

The 68000 Instructions:

First of all the basics. Anything after a semicolon (“;”) is a comment and not a command. Our Assembler will ignore all comments, just as if they had never been typed in. Each line of code has three basic parts¹¹. If you look at the code you can see that it seems to be divided into three columns. The first column begins with the first character of each new line and is called a *label*. Labels are used so we can refer to that line of code from within our program. Anything in the second column is an instruction for the 68000 (or the 68881). Anything in the third column tells the chip how (and where) to execute that instruction.

A couple of the instructions that appear at the beginning of the program are actually instructions for the Assembler¹² and not the chip. The instruction “EQU” as in “ReturnAdd EQU 4” tells the Assembler to substitute the number “4” every time it sees the label “ReturnAdd” in our code. We could have written the number 4 in ourselves, but using the label “ReturnAdd” makes our code a lot more understandable¹³. FUNC tells the Assembler that this is the name of a function. The word EXPORT tells it to let the rest of the world know that a function exists by the name “ScaleMult” and where that function can be found. MC68881 states that we will be using some of the 68881 instructions (otherwise our Assembler would say we must be making a mistake when we try to use them). MACHINE MC68020 lets the Assembler use instructions that are available only on the 68020 chip on a Mac II. The MPW Assembler will replace some of our instructions with it's own if it sees a faster way of doing the same thing. This is called optimizing the code. If you are not running with a 68020 delete this line. OPWORD is similar to

EQU except that it lets us define a new instruction by its numeric equivalent, whereas EQU lets us define a constant used by the instruction. ENDF and END signal that we are finished writing our function and code, respectively.

The 68000 chip has several registers for holding numbers. These can be thought of as Mail Box slots if you will. A mail box can either hold a parcel (your data) or a slip telling you where your parcel is (an address). Our registers take 32 bit numbers and hold them on the chip. Eight slots (or “registers”) hold simple data. Another eight registers hold 32 bit addresses (see Figure 1). Address are simply numbers that tell the Mac which byte in the memory holds your data. 0 is the first byte in RAM, 1 is the second, etc. Remember that both Data and the Address registers hold simple 32 bit numbers. These registers are referred to as D0, D1, ..., D7 for the data and A0, A1, ..., A7 for the Addresses. To manipulate numbers on the computer we simply move a number from RAM, into one of these registers, do what we want to it, then put it back. The fact that the 68000 CPU has so many registers on the chip means that we don't have to swap back and forth between RAM and the chip nearly as often as you had to on older chips. Our basic command for moving data around is “MOVE” (doesn't that make sense?).

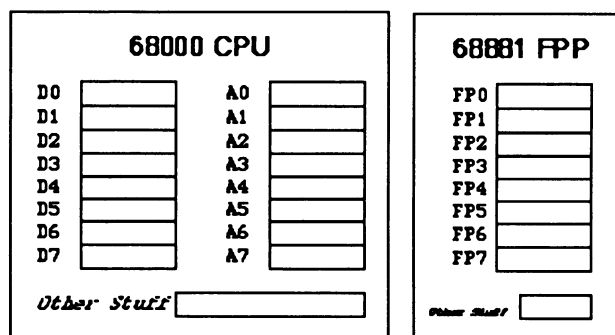


Figure 1. The Microprocessors

To get a number that is stored in the 128th byte of RAM, and move it to register D1 on our chip would simply write “MOVE 128,D1” (see Figure 2). To actually place the number 68 into register D1 we would say “MOVE #68,D1” (see Figure 3). The symbol “#” means to take the number and use it literally, versus using the number as the address of our data, as we showed in Figure 2.

Putting the name of a register in parentheses means to use the address stored within that register to find our data. This is shown in Figure 4. The same instruction without the parentheses means to use the contents of the register directly, as shown in Figure 5.

Basic 68881 Commands

Notice that many of the instructions have suffixes attached to them, such as MOVE.W or MOVE.L (and MOVE.B). These simply tell the 68000 what size number we are moving, “.B” for a Byte (8 bits long), “.W” for a Word (16 bits long and the default if you forget to specify a suffix), or finally “.L” for a Long word (or 32 bits). Some of the Floating Point instructions specify a “.X” which tells the 68881 to move three long words (or the 96

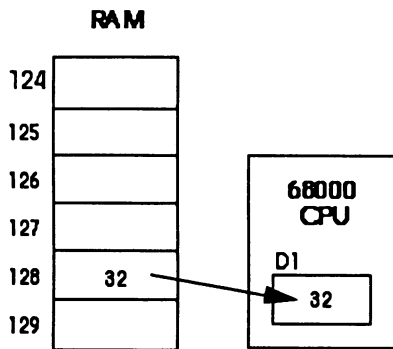


Figure 2. MOVE 128, D1
(get the data from memory location 128)

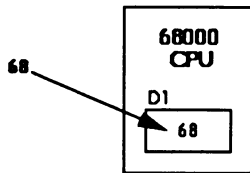


Figure 3. MOVE #68, D1
(put 68 in register D1)

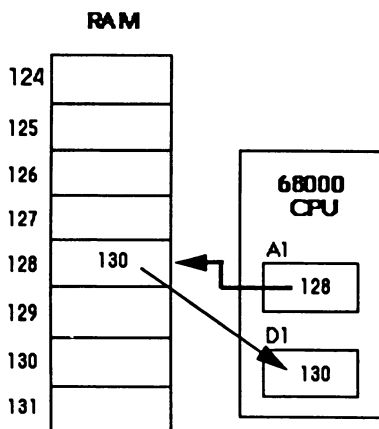


Figure 4. MOVE (A1), D1
(register A1 contains the address of our data)

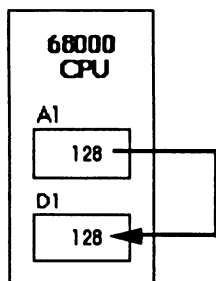


Figure 5. Move A1, D1
(register A1 contains the data)

bits it takes to specify an extended number). The ".X" extension is only understood by the 68881, so if you added it to a 68000 instruction, the CPU would not know what you were talking about. When we move different size numbers onto the chip they are always placed so that bit zero is aligned all the way to the right. Thus the three numbers would be stored as in Figure 9.

The instruction **FMOVE** works the same way as **MOVE**, but allows you to move numbers to and from the 68881 coprocessor chip. By using the suffix ".X" (as in **FMOVE.X**) we can tell the coprocessor that you are manipulating a 96 bit extended number, although ".L", ".W", ".B" will also work if you are referring to these quantities. The 68881 FPP has 8 registers of its own (FP0 to FP7), each with enough room to store an extended number.

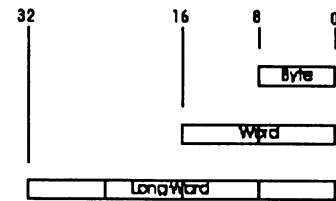


Figure 6. How Numbers Are Stored in the 68000 Registers

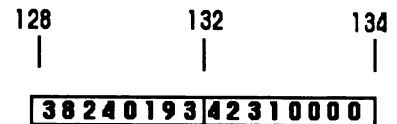


Figure 7. How numbers are stored in RAM

Bits and Bytes

I am not going to go over a lengthy explanation of bits, bytes, or how to convert between decimal numbers and hexadecimal numbers (since these are just different ways of writing the same number). There are many good books on these kinds of basics, and they are referred to in the appendix ^[1-3].

The 68000 stores numbers in memory in the same way that you would read them from a piece of paper. It starts at a low memory address and writes the number, starting with the most significant digit and going to the least significant digit. If another number is to be written, it will repeat the sequence. Thus if I was going to store the number 38240193 at memory location 128, and then store the number 42310000 in the next highest position (which would start at memory location 132) the memory would appear as in Figure 7.

The stack is a key concept to writing assembly language. The best example of a stack are those old plate stackers that we all remember from the school cafeteria. If you imagine that each plate is a number, you simply place the next "plate" on the stack, piling them up as you go. If you need a plate, you simply pop it off the top. The 68000 uses the same concept to store numbers. To define a stack you first pick where in memory you want to start. This starting address is usually stashed in register A6 (this is simply a convention and we could just as easily stash it in A4 or any other register). We then place a number on the stack (we call this pushing the number onto the stack) by simply writing the number into memory, using the memory location pointed to by register A6. The next thing we want to keep track of is the end of our stack. We can do this using another one of our registers, A7. If we just added 2 bytes to the stack by pushing our number, then we need to adjust register A7 to point to the next free byte of memory, or in other words, the end of the stack.

What if we want to take the top number off of the stack? A7 points to the top, so we get our number, move it somewhere else, then change A7 to point to the new top of the stack. Using this system we can push numbers onto the top of the stack, and pop them off at will. This type of memory arrangement is called a Last In/First Out system, since the last number we pushed onto the stack is the First

number we have available to pop off again.

Most computer systems use the stack as a method of storing numbers. The only little twist to the way the Macintosh handles a stack is that instead of starting at a low memory address and working higher, the Macintosh starts at a High memory address and works downward, toward lower memory. This twist doesn't change the way we work with the stack. If we decide that we want to start our stack at memory address 128, we store 128 in register A6 and then set A7 to point to the end of the stack, which in this case is also 128. We now want to add a 4 bytes number onto the stack, such as \$32411000. We simply subtract 4 from the address in A7 (or $128 - 4 = 124$), and Move our 4 byte number to RAM, starting at address 124. This is accomplished with the command "MOVE.L #\$32411000, -(A7)". If we look at the RAM, starting at address 124 we will now see the number \$32411000.

Take a close look at the instruction we just issued. By appending .L to the MOVE command we have told the assembler that we are moving a long word, which is by definition 4 bytes long (or 32 bits). The #\$32411000 portion used tells the 68000 to use the number \$32411000 literally. Finally the command "-(A7)" does something special. It tells the 68000 to subtract the length of our longword, 4, from the address stored in A7, and place the resulting address back in A7. It also does this *before* it executes the MOVE instruction, so our number gets moved to address 124 instead of 128. This leaves 124 in A7 when we are finished so our stack pointer now points to the end of the stack again. We call this kind of addressing "Predecrement Indirect". Don't worry about what this mode of addressing is called, just take note of what it does. The fact that the 68000 can do these kinds of instructions in one step makes life very simple for us. The 68000 can also do the converse operation. Lets say we want to remove our number and store it in RAM starting at address 200. We will then issue the command "MOVE.L (A7)+, 200". Again the suffix ".L" tells us that we are dealing with a 4 byte longword. Note that if we used ".W" instead the 68000 would do the same calculation, but know that we were talking about moving a (2 byte) *word* instead. Our original command, MOVE.L, will get the 4 byte number, starting at the memory location pointed to by register A7, and move that number to memory address 200. After it is done, it will add 4 to the address stored in register A7 so that A7 now contains $124 + 4$ or 128, which is the top of the stack again. This type of addressing is called Postincrement Indirect. We have just successfully pushed a 4 byte number onto the stack, and then popped it off again, all the time keeping register A7 pointing to the end of the stack.

We can use this algorithm to push many numbers onto the stack, one after another. This is exactly what a Pascal program does when it calls your assembly language subroutine. It pushes all the variables it is going to hand you onto the stack, and then jumps to your routine. Since we know the kinds of numbers that the main program will pass to us, we can pop them off, do our calculation, push our result back onto the stack, then jump back to the main program. We will talk more about actually doing this a little later.

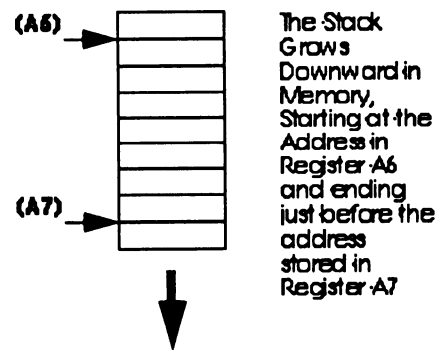


Figure 8. The Macintosh Stack

More Instructions

Let's look at a few more Assembly language instructions. The instruction CLR (which stands for Clear) will put all zeros at an address. Thus "CLW.W 128" will put 2 bytes worth of zeros starting at address 128. CLW.L (A1) will store 4 bytes worth of zeros at the address stored in register A1. CLW.B D1 will fill the 1st byte of register D1 with zeros.

The instruction ADD will do what it's name would suggest. "ADD.L D1,D2" will add the two long words stored in registers D1 and D2 and store the result in D2. "ADD.W (A1)+,D1" will get a number from memory, stored at the address contained in A1. It will then add that number to the number stored in D1. The final result is stored in D1. The fact that we wrote "(A1)+" means that after it completes all of this addition, it will increase the address stored in A1 by 2 (since we specified .W). This is very useful if we are adding a bunch of numbers to D1 and they are stored in memory one after another starting at the address in A1. After we execute our "ADD (A1)+,D1" command, A1 now automatically points to the next number in memory. This command, when placed in a simple loop, is a powerful way to add many numbers.

Another use for this type of addressing to move a range of bytes from one part of RAM to another. If we store the beginning address of our data in A1, and the address of the place we want to move it to in A2, by repeatedly calling "MOVE.L (A1)+, (A2)+" we move our data from (A1) to (A2) in RAM, afterwards both the addresses stored in A1 and A2 are automatically incremented by 4, so we are ready to move the next long word. Nifty, isn't it?

The commands LSR and LSL stand for Logical Shift Right and Logical Shift Left. What they do is move each bit in memory to the right or left a certain number of places. The bits that fall off the end disappear¹⁴, and the places that we leave empty are replaced by zeros. An example of a Logical Shift Left is shown in Figure 9. "LSL.L #5,D1" will shift the binary number stored in register D1 to the left 5 places by pushing zeros onto the right. A LSR instruction works in the converse fashion

LSR and LSL will become important later on when we want to quickly manipulate our numbers. The next instruction that we use a bit is LEA which stands for *Load Effective Address*. An example might be "LEA Scalar(A6),A2". This instruction calculates the address using the code 16(A6) and stores that address in register A2 (remember that we had defined the label Scalar to

be the same as the number 16). Note that it doesn't go to the address, just calculates what it should be. Unfortunately you probably don't know what 16(A6) would be, but it is not hard to figure out. We take the address stored in register A6 (lets say it's our old friend 128 again) and add 16 to it. Take the result ($128 + 16 = 144$) and put it in A2. We can now use the address in A2 for other instructions. This type of instruction is good if we know we want a number stored 16 bytes away from some base address, and that base address will be given to us later. When we get that base address, we can store it in A6 and use our LEA instruction to find the address of our data. Many times we will know the form a bunch of data, i.e. an integer is stored first, then a real number, then a pointer to an array (i.e. the address of some other data), but we won't know exactly where this data is going to be in RAM. LEA lets us get the beginning of our data and calculate where

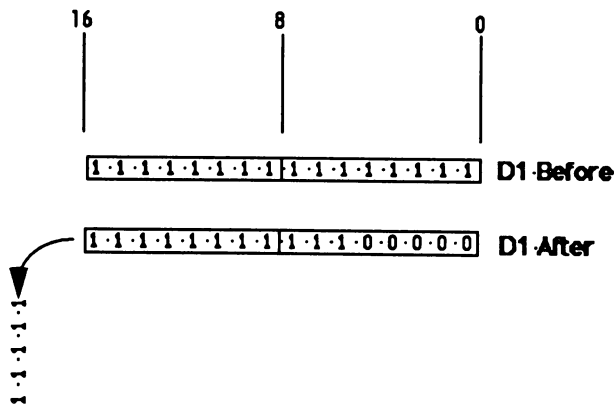


Figure 9. LSR #5, D1

everything else should be.

The final instruction that we need to look closely at are the LINK and UNLK instructions (i.e. "LINK A6,#-12" & "UNLK A6"). If you remember when we were talking about stacks, we said we would remember the beginning address of the stack by putting it in register A6. The technique of beginning your stack at A6 is called *stack frames*. The way it works is as follows: The main Pascal program has it's own stack. It pushes all of the data it needs to give your subroutine on it's stack. Finally it pushes a return address onto the stack (so you know which instruction to return to when you're done executing your code), and then jumps to your assembly code. You dutifully save the return address by popping it off the stack and moving it someplace safe. You then pop all of the data off the stack and are ready to go.

At this point you most likely want to create your own stack, and the best place is right at the end of the main program's stack. The first thing you do is save the old value of A6 that marks the beginning of Pascal's programs stack. A great way to store this value is to push it right onto the stack. Take the new value of A7 (the address that we are going to use as the base of our new stack) and store that in A6. We are now ready to go. A6 and A7 both point to the beginning of our stack. We then increment the stack pointer (A7) to make room for any storage space that you might want to use. As we push and pop values to and from our stack we will adjust A7 appropriately with the predecrementing and

postincrementing commands that we discussed. These steps are illustrated in Figure 10. All of the steps necessary to create a stack frame are contained in the LINK instruction. The instruction "LINK A6,#-12" will first push A6 onto the stack after decreasing A7 by 4 (since an address is a long word of 4 bytes). The new new value of A7 is copied into A6, since that will be the base of our new stack. Finally the stack pointer A7 is decremented by 12 bytes to make room for new data that we might want to store there. In this case we decremented the stack by 12 since, if you remember, our stack grows downward in memory. Decreasing A7 by 12 leaves room for 12 new bytes of storage space (which would be just enough room for us to stash a 96 bit extended

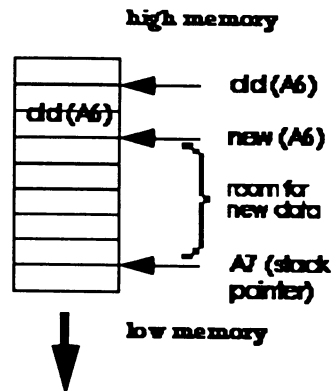


Figure 10. The Stack After LINK

number in the 68881 format!).

The nice thing about having A6 point to the beginning of our stack is that when we are ready to jump back to the main program we can take A6 and copy it into A7. That maneuver has just made our stack pointer point to the bottom of our stack frame. We then pop the old value of A6 off the stack and put it back into register A6, then add 4 to the contents of A7, just like our old friend the "MOVE (A7)+,A6" instruction. We now have returned the stack to it's state just before we created our stack frame. The 68000 does all of this with another single instruction UNLK. In our case "UNLK A6" will do all of the above steps.

The second nicety of having A6 point to the bottom of our stack frame is that we can easily store our own variables on the stack. Although we do not know in advance where A6 will point, we do know the relative position of our data on the stack. Let's say one variable will be located 4 bytes before A6, another will be 6 bytes before A6, etc. By using addresses of the form -4(A6) and -6(A6) we can access these variables whenever we need to.

Another advantage of addressing all of our data relative to the stack frame in (A6), is that we don't necessarily have to waste time moving the variables that we received from our Pascal, off of the stack. Since we know what the stack looks like after we execute our "LINK A6,#-12" instruction (see Figure 10) we know that the pointer to inMat is located 8 bytes prior to (A6), and that outMat will be 12 bytes before (A6). We then use the same form of addressing (e.g. "MOVE.L 12(A6),A0") to move outMat to A0 so we can work with it.

In order to quickly manipulate every element in our matrix,

we will create a loop that performs our calculation, then repeats itself with the next element in the matrix, until we get to the end. To create a loop and test for a specific condition (so we know when to stop looping) we will use the instruction DBLT (decrement and branch if less than zero). This is one member of a class of instructions written in shorthand as DBcc, where cc is a test. The easiest way to think of this instruction is *don't* branch if the condition is true. The instruction works by first testing if the test condition is true, i.e. did we just execute an instruction that resulted in an answer that was less than zero (or in other words a negative number). If the prior instruction created a negative number then don't branch (which would then leave our loop, just as branching would bring us back to the beginning of our loop), but instead execute the next instruction. Now if the last instruction created a positive number (and thus the test failed) then we decrease the register D1 by one and see if D1 is now less than zero. If it is, then we *don't* branch but instead continue on to the next instruction. If neither of these condition are true then round we go again, branching to the beginning of our loop. This instruction has several advantages. First it looks for a specific condition that will end the loop. Failing this, it counts down a register until it is less than zero and then exits the loop.

The last instruction in our program is RTS (Return from Subroutine). This instruction assumes that we have pushed the return address back onto the top of the stack and will jump to that address. If we had stored our return address in A0 we could have accomplished the same feat by using the command "JMP (A0)" (jump to the address stored in register A0).

SANE vs. the 68881

Although SANE and the 68881 both use the same basic format for an extended number, there is one fundamental difference. In SANE the extended number is 80 bits (10 bytes) long. On the other hand, the 68881 places two bytes of zeros just before the last word in the number (this is illustrated in Figure 11). The 68881 does this because it is more efficient for it to move three long words than to manipulate 2 1/2 long words. We can easily convert back and forth between these formats by grabbing the last word of a SANE extended number and shifting it over 16 spaces to the left (using the LSL instruction). When we want to return the number to SANE we simple shift the top long word of our 68881 extended number back to the right 16 spaces (using

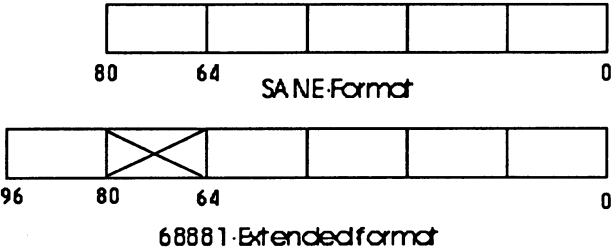


Figure 11. SANE vs. 68881 extended numbers

LSR)¹⁵.

The Interface for a Pascal calling routine

A Pascal program calls a procedure in a very orderly fashion. In our example the procedure that we will write in assembly language is declared to Pascal as:

```
FUNCTION ScaleMult (scalar : extended;VAR inMat, outMat :
matrix) : integer;
external;
```

Pascal will prepare to call our subroutine by first pushing enough space onto the stack for our result. If our subroutine was declared as a *procedure*, Pascal would not expect a return value and would skip this step. Following this, Pascal pushes it's data onto the stack as read from left to right. In our case Pascal would next push a pointer to the variable "scalar". Pascal pushes a pointer¹⁶ to the data if it is declared as a VAR parameter (meaning that the subroutine can change the actual variable), or if the data is larger than four bytes in size¹⁷. A pointer to inMat is then pushed onto the stack followed, by a pointer to outMat. Finally the return address is pushed onto the stack and the program then jumps to our subroutine. Pascal cannot do any type checking when it jumps to an assembly language routine. It simply assumes you know what kind of data you are expecting, and that you will place the correct result on the stack (if your routine is a function and not a procedure) when you return to the main program.

When the Pascal calling routine tries to make space for the return variable (the very first thing we said that Pascal would push onto our stack if it is calling a function) it follows the Pascal rule of pushing the actual variable if it is 4 bytes or less, or, as is the case of our extended variables which is larger than 4 bytes, it pushes a pointer to the variable. We would then use that address to store the result of our functions. When we return to Pascal we would leave the address of our return variable on the stack so the calling Pascal program can remember where to get its result.

In our assembly language program we will remove all the information from the stack, complete our routine, then push an integer (2 bytes) onto the stack as a result. Our final maneuver will be to jump back to the return address that was given to us initially. Figure 12 shows us the stack as it would appear when we initially enter our subroutine.

high memory	
return value	low word
	high word
pointer to scalar	low word
	high word
pointer to inMat	low word
	high word
pointer to outMat	low word
	high word
return address	low word
	high word
low memory	

Figure 12. The Stack as passed from Pascal

One point I would like to clarify is this business of Pascal pushing a pointer onto the stack if that variable is more than four bytes long, or if it is declared as a VAR in the interface statement. Figure 13 shows the size of some common variables used in Pascal.

<u>Variable</u>	<u>What is passed on the stack</u>
integer	2 bytes
longInt	4 bytes
real	4 bytes
extended	4 byte pointer to the 10 bytes SANE extended number
pointer	4 bytes
string	4 bytes pointer0 to your string if it is over 4 bytes in size) A string takes up n+1 bytes where n is the number of characters. The very first byte tells you how many characters are in the string.
boolean	2 bytes (put your result in the least significant byte of the word when you return to Pascal)
char	2 bytes
anything	4 byte pointer to the variable in RAM declared as "var"

Figure 13. Size of Pascal Variables Pushed onto the stack

Remember, if we wrote a procedure instead of a function we would leave the stack empty when we returned to the main program. An illuminating example would be if we had declared our function to return an extended number. Since we don't want to push the entire 10 bytes of data onto the stack we simply store our data at the address supplied by the calling routine. I always considered it courteous of Pascal to take care of the memory for any variables our function needs to return.

In our program, we are returning an integer which is two bytes in size, so we can just push the actual value of the integer onto the stack, and not worry about pointers at all. Remember that Pascal has already made enough room on the stack to hold our integer.

Pascal lets us freely use registers A0, A1, D0, D1, and D2. We can of course use any of the registers on the 68000 chip, just so long as we save the values that are stored there and put them back, before we return to the Pascal environment.

The Code

Now that we understand all the the basic instructions, lets go through our assembly language code. Our program receives three pointers (or addresses) from Pascal. The data is stored at these addresses in SANE extended number format. We take each of these numbers, and shift over the top word so it is in the correct format for the 68881 chip. Next, we move one element of our matrix (from inMat) onto the 68881, along with the scalar, multiply them, then save the result in the output matrix (outMat).

We repeat this for all of the numbers in our matrix before returning to the main program. The LINK instruction is used to illustrate how we can make room on the stack for our own variables, easily accessing them relative to the base of the stack. Our funny data structure for an element may make a little more sense now, in light of the above differences between the SANE and 68881 representation of an extended number. Each element of our vector is a record of the form

```
element = PACKED RECORD
    empty : integer;
    n : extended
END;
```

the empty integer that I have thrown before each SANE extended number takes up one *word* of space (16 bits). This gives us enough room to change all of our 80 bit SANE extended numbers into the larger 96 bit 68881 format. Because we have this extra room we can store our numbers right back in their array. This may not be important for our simple little program, but if we were doing a lot of math with these numbers, converting back and forth for each mathematical operation would take a lot of time. This way we can convert each element to the 68881 extended format when we start our program and convert them all back to the SANE format when we are finished.

A vector is simply defined as an array of many elements.

```
vector = ARRAY[0..19] OF element;
matrix = RECORD
    rows : integer;
    columns : integer;
    vecPtr : ^vector
END;
```

A matrix is a data structure that specifies the number of rows and columns it will contain, followed by a vector that is rows * columns in size. Again, all of this is not strictly needed for our code, but it does illustrate how we can write a general algorithm that would take an matrix of an arbitrary size, do whatever math is needed, and then store the results in a new matrix.

How to Compile and Put the Program Together

I have always hated articles that give you great code¹⁸, but then leave you in the lurch as to *exactly* how they put it all together. I'm going to give instructions for using the MPW compiler, since it already supports the 68881 and 68020 chips.

First of all open the MPW shell and type Command-N to get a fresh document. Type in all of the Assembly code, as it is written (of course you can leave out the comments if you want). Save this document as "ScalarMult.a". Click on the Worksheet¹⁹ and type the Magical incantation "Asm ScalarMult.a" followed by the enter key (or Command-RETURN if you don't have an enter key on your keyboard). [Please note that the *return* key and *enter* key are treated differently in the MPW environment.] If you didn't make any mistakes then MPW will have just created a file named "ScalarMult.a.o". If everything worked out alright then quit MPW.

Version 1.11 of LightSpeed Pascal includes a little utility called the “**O converter**”; Double click on this icon, then select our assembly code file “ScalarMult.a.o” from the dialog box. After you quit “**O converter**”, You will see that we have just created a LightSpeed library file.

Our final chore will be to launch LightSpeed Pascal and open a new project. First get a fresh window up, and type in the Pascal main program. Next go to the “**Project**” Menu and select “**Add Files...**”. Add the main program that we just typed in, along with the library file we created from our assembly code. That’s all there is to it!. You can either run the program by typing **Command-G** (for Go) from within Lightspeed, or elect to build a stand alone program²⁰.

Remember that our procedure will jump into **Macsbug** as the first instruction, so that we can follow the program. If you don’t have **Macsbug** in your system folder, remove the instruction “_Debugger” or the computer will bomb with an **ID=1** error

Summary

In this article, I hope I have introduced enough basic information to let you write an assembly language subroutine. We looked at the basic 68000 instructions, and a few instructions from the 68881 numeric coprocessor chips. As we saw 68881 instructions are easily executed, as if they were part of a souped up 68000 CPU.

We saw how Pascal calls functions and procedures and we created a short assembly language subroutine that could store its own local variables and manipulated them. Finally we looked at how to compile and interface the assembly language using Light Speed Pascal and the MPW assembler. I hope that this introduction lets you see that assembly language is not as intimidating as it looks, and that by selectively rewriting certain Pascal procedures we can realize great increase in the speed of many of our programs, especially those that are calculation intensive.

Footnotes

1. Apples “Standard Apple Numeric Environment” which handles complex math very well, if not a little slowly

2. Note that in many cases we are presenting a simplified picture of what our Assembly language instructions do. If you are an experienced Assembly language programmer, you may note some deviations from fact (i.e. white lies). This is to shield the inexperienced user with unnecessary, complicating details, and will be noted. Come to think of it, if your such a hot shot programmer, why do you need to read this? See any of the books in the reference section for a more complete picture of Assembly Language programming.

3. FPP

4. Central Processing Unit.

5. I know... when you ask the salesman, they’ve never heard of it.

6. Macintosh Programmers Workshop

7. And the only native Mac language system supported by Apple for a long time

8. Yes, I know the “definition” of AssUMe.

9. If you didn’t know it, you do now.

10. A hexadecimal number is a number, similar to our ordinary base 10 counting numbers. Instead of counting from zero to nine (then indicating ten by putting a one in the next significant place, as in 10), in hexadecimal we count from zero to fifteen (representing the numbers ten through fifteen with the symbols A through F). We would then represent the next number, 16, by putting a one in the next significant place as in \$10. I will use the “\$” sign before any hexadecimal number to you don’t confuse \$10 with the decimal number ten. Hexadecimal notation is a convenient way to write numbers when we deal with computers. See any of the books in the reference section for a good review of the subject.

11. Not counting the comments

12. The program that translates our written words into the actual numbers that the 68000 and 68881 understands

13. At least for us humans. The computer prefers numbers and will substitute numbers for all of our words, including the instructions. Oh well... to each his own.

14. Not exactly, each bit first goes to a place called a Carry Bit, before it drops off into never never land, but that’s not really important.

15. An even easier way to accomplish the same thing would be to MOVE the last word over, for example using the instruction “MOVE 8(AØ), 1Ø(AØ)”, provided AØ pointed to the beginning of our extended number. You should always follow the above instruction with CLR.W 8(AØ) to ensure that the bit 64 through 80 are zero, just to ensure compatibility with future products.

16. A pointer is simply the address of where our data starts in RAM

17. Note that the pointer given to us actually points to a copy of the variable, known as a “dummy variable”. This will prevent us from accidentally changing the value of the variable in the main program.

18. Don’t worry, I’m not assuming that you have gotten any great code from this article

19. Its the document that doesn’t have a go-away box

20. Of course you could have complied and linked you program totally from within MPW and MPW Pascal, but if you know how to do all of that, you probably didn’t need my help to begin with.

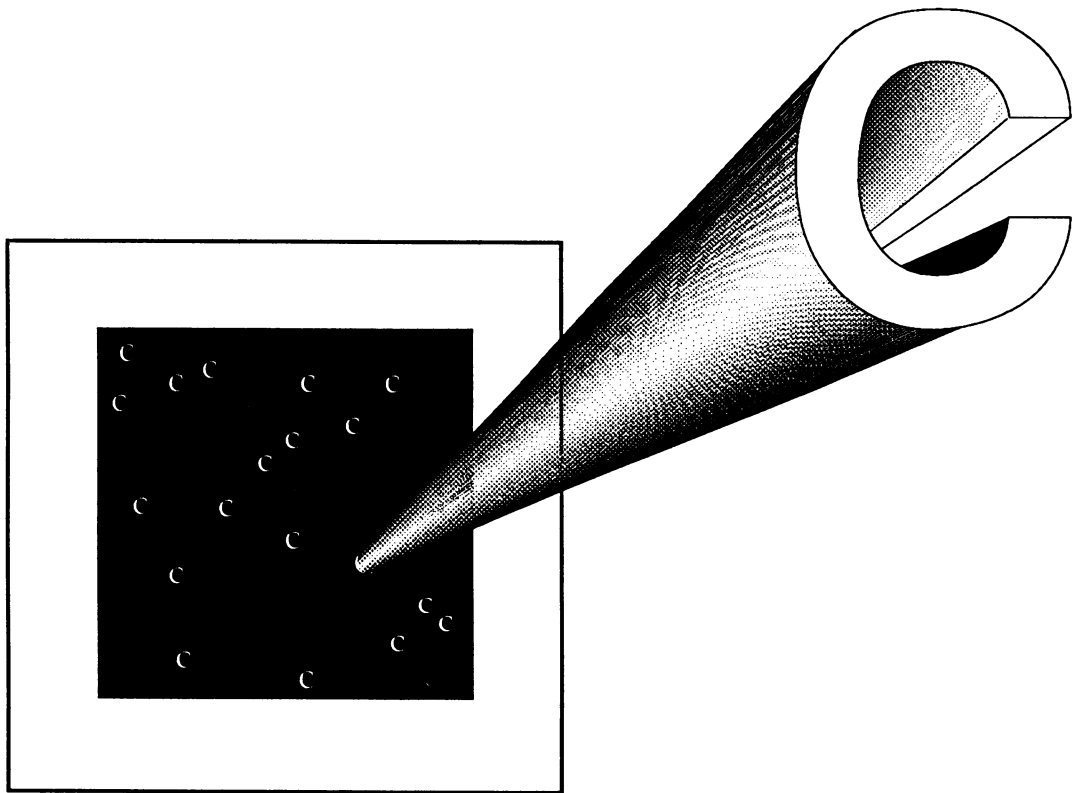
Bibliography

[1] *MPW and Assembly Language Programming for the Macintosh*, Scott Kronick, Hayden Books, 1987

[2] *Programming the Macintosh in Assembly Language*, James Coffron, Sybex Inc., 1986

[3] *How to Write Macintosh Software*, Scott Knaster, Hayden Book Company, 1986





C Workshop

C Workshop

Bezier Curve Ahead!



David W. Smith
Los Gatos, CA
Volume 5, Number 1

David W. Smith (no known relation to the Editor) is a Sr. Software Engineer at ACM Research, Inc., in Los Gatos.

There comes a time in the development of some applications when arcs and wedges just don't cut the mustard. You want to draw a pretty curve from point A to point B, and QuickDraw isn't giving you any help. It seems like a good time to reach for a computer graphics text, blow the dust off of your college math, and try to decipher their explanation of splines. Stop. All is not lost. The Bezier curve may be just what you need.

Bezier Curves

Bezier curves (pronounced "bez-yeah", after their inventor, a French mathematician) are well suited to graphics applications on the Macintosh for a number of reasons. First, they're simple to describe. A curve is a function of four points. Second, the curve is efficient to calculate. From a precomputed table, the segments of the curve can be produced using only fixed-point multiplication. No trig, no messy quadratics, and no inSANEity. Third, and, to some, the most important, the Bezier curve is directly supported by the PostScript curve and curveto operators, and is one of the components of PostScript's outlined fonts. The Bezier curve is also one of the principle drawing elements of Adobe Illustrator™. (Recently, they've shown up in a number of other places.)

Bezier curves have some interesting properties. Unlike some other classes of curves, they can fold over on themselves. They can also be joined together to form smooth (continuous) shapes. Figure 1 shows a few Bezier curves, including two that are joined to form a smooth shape.

The Gruesome Details

The description of Bezier curves below is going to get a bit technical. If you're not comfortable with the math, you can trust that the algorithm works, and skip ahead to the implementation. However, if you're curious about how the curves work and how to optimize their implementation, or just don't trust using code that you don't understand, read on.

The Bezier curve is a parametric function of four points; two endpoints and two "control" points. The curve connects the endpoints, but doesn't necessarily touch the control points. The general form Bezier equation, which describes each point on the curve as a function of time, is:

$$P(t) = w_1(t)P_1 + w_2(t)P_2 + w_3(t)P_3 + w_4(t)P_4$$

where P_1 and P_4 are the endpoints, P_2 and P_3 are the control points, and the w_n 's are weighting functions, which blend the four points to produce the curve. (The weights are applied to the x and y components of each point independently.) The single parameter t represents time, and varies from 0 to 1. The full form of the Bezier curve is:

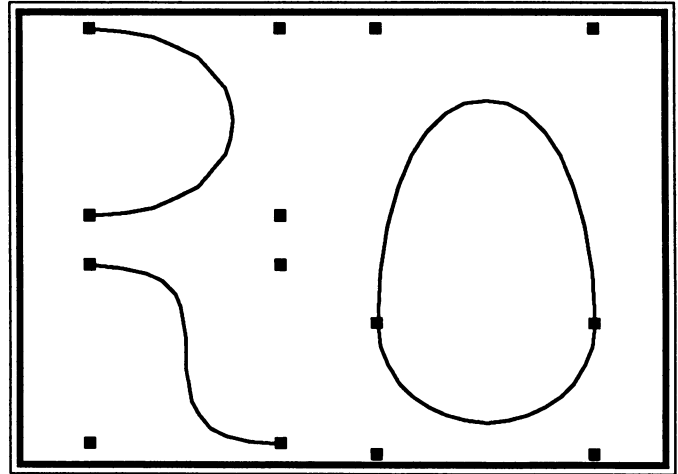


Figure 1. Some Belzer Curves and Shapes

$$P(t) = (t-1)^3P_1 + 3t^3(t-1)P_2 + 3t(1-t)^2P_3 + t^3P_4$$

We know that the curve touches each endpoint, so it isn't too surprising that at $t=0$ the first weighting function is 1 and all others are 0 (i.e., the initial point on the curve is the first endpoint). Likewise, at $t=1$, the fourth weighting function is 1 and the rest are 0. However, it's what happens between 0 and 1 that's really interesting. A quick side-trip into calculus to take some first derivatives tells us that the second weighting function is maximized (has its greatest impact on the curve) at $t=1/3$, and the third weight is maximized at $t=2/3$. But the clever part—the bit that the graphics books don't bother to mention—run the curve backwards by solving the equation for $1-t$, and you find that $w_1(t)=w_4(1-t)$ and $w_2(t)=w_3(1-t)$. As we'll see below, this symmetry halves the effort needed to compute values for the weights.

Implementing Bezier Curves

One strategy for implementing Bezier curves is to divide the curve into a fixed number of segments and then to pre-compute the values of the weighting functions for each of the segments. The greater the number of segments, the smoother the curve. (I've found that 16 works well for display purposes, but 32 is better for hardcopy.) Computing any given curve becomes a simple matter of using the four points and the precomputed weights to produce the end-points of the curve segments. Fixed-point math yields reasonable accuracy, and is a hands down winner over SANE on the older (pre-Mac II) Macs, so we'll use it.

We can optimize the process a bit. The curve touches each endpoint, so we can assume weights of 0 or 1 and needn't compute weights for these points. Another optimization saves

both time and space. By taking advantage of the symmetric nature of the Bezier equation, we can compute arrays of values for the first two of the weighting functions, and obtain values for the other two weights by indexing backwards into the arrays.

Drawing the curve, given the endpoints of the segments, is the duty of QuickDraw (or of PostScript, if you're really hacking).

The listing below shows a reasonably efficient implementation of Bezier curves in Lightspeed C™. A few reminders about fixed-point math: an integer times a fixed-point number yields a fixed-point number, and a fixed by fixed multiplication uses a trap. The storage requirement for the algorithm, assuming 16 segments, (32 fixed-point values), is around 32*4*4, or 512 bytes. The algorithm computes all of the segments before drawing them so that the drawing can be done at full speed. (Having all of the segments around at one time can be useful for other reasons.)

More Fun With Curves

Given an implementation for Bezier curves, there are some neat things that fall out for almost free. Drawing a set of joined curves within an **OpenPoly/ClosePoly** or an **OpenRgn/CloseRgn** envelope yields an object that can be filled with a pattern. (Shades of popular illustration packages?) For that matter, lines, arcs, wedges, and Bezier curves can be joined to produce complicated shapes, such as outlined fonts. Given the direct mapping to PostScript's **curve** and **curveto** operators, Bezier curves are a natural for taking better advantage of the LaserWriter.

As mentioned above, Bezier curves can be joined smoothly to produce more complicated shapes (see figure 1). The catch is that the point at which two curves are joined, and the adjacent control points, must be *colinear* (i.e., the three points must lay on a line). If you take a close look at Adobe Illustrator's drawing tool, you'll see what this means.

One nonobvious use of Bezier curves is in animation. The endpoints of the segments can be used as anchor-points for redrawing an object, giving it the effect of moving smoothly along the curve. One backgammon program that I've seen moves the tiles along invisible Bezier curves, and the effect is very impressive. For animation, you would probably want to vary the number of segments. Fortunately, the algorithm below is easily rewritten to produce the *n*th segment of an *m* segment curve given the the end and control points.

Further Optimizations

If you're really tight on space or pressed for speed, there are a few things that you can do to tighten up the algorithm. A bit of code space (and a negligible amount of time) can be preserved by eliminating the setup code in favor of statically initializing the weight arrays with precomputed constant values. Drawing can be optimized by using **GetTrapAddress** to find the address in ROM of **lineto**, and then by calling it directly from inline assembly language, bypassing the trap mechanism. I've found that neither optimization is necessary for reasonable performance.

```

/*
** Bezier - Support for Bezier curves
** Herein reside support routines for drawing Bezier curves.
** Copyright (C) 1987, 1988 David W. Smith
** Submitted to MacTutor for their source-disk.
*/

#include <MacTypes.h>
/*
    The greater the number of curve segments, the smoother the
    curve, and the longer it takes to generate and draw. The
    number below was pulled out of a hat, and seems to work o.k.
*/
#define SEGMENTS 16

static Fixed weight1[SEGMENTS + 1];
static Fixed weight2[SEGMENTS + 1];

#define w1(s) weight1[s]
#define w2(s) weight2[s]
#define w3(s) weight2[SEGMENTS - s]
#define w4(s) weight1[SEGMENTS - s]

/*
 * SetupBezier - one-time setup code.
 * Compute the weights for the Bezier function.
 * For the those concerned with space, the tables can be
 * precomputed. Setup is done here for purposes of illustration.
*/
void
SetupBezier()
(
    Fixed      t, zero, one;
    int        s;

    zero = FixRatio(0, 1);
    one = FixRatio(1, 1);
    weight1[0] = one;
    weight2[0] = zero;
    for ( s = 1 ; s < SEGMENTS ; ++s ) (
        t = FixRatio(s, SEGMENTS);
        weight1[s] = FixMul(one - t, FixMul(one - t, one - t));
        weight2[s] = 3 * FixMul(t, FixMul(t - one, t - one));
    )
    weight1[SEGMENTS] = zero;
    weight2[SEGMENTS] = zero;
)

/*
 * computeSegments - compute segments for the Bezier curve
 * Compute the segments along the curve.
 * The curve touches the endpoints, so don't bother to
 * compute them.
*/
static void
computeSegments(p1, p2, p3, p4, segment)
    Point      p1, p2, p3, p4;
    Point      segment[];
(
    int        s;

    segment[0] = p1;
    for ( s = 1 ; s < SEGMENTS ; ++s ) (
        segment[s].v = FixRound(w1(s) * p1.v + w2(s) * p2.v +
                                w3(s) * p3.v + w4(s) * p4.v);
        segment[s].h = FixRound(w1(s) * p1.h + w2(s) * p2.h +
                                w3(s) * p3.h + w4(s) * p4.h);
    )
    segment[SEGMENTS] = p4;
)

/*
 * BezierCurve - Draw a Bezier Curve
 * Draw a curve with the given endpoints (p1, p4), and the

```

```

given
* control points (p2, p3).
* Note that we make no assumptions about pen or pen mode.
*/
void
BezierCurve(p1, p2, p3, p4)
    Point    p1, p2, p3, p4;
{
    int      s;
    Point    segment[SEGMENTS + 1];

    computeSegments(p1, p2, p3, p4, segment);
    MoveTo(segment[0].h, segment[0].v);

    for ( s = 1 ; s <= SEGMENTS ; ++s ) {
        if ( segment[s].h != segment[s - 1].h ||
            segment[s].v != segment[s - 1].v ) {
            LineTo(segment[s].h, segment[s].v);
        }
    }
}

/*
** CurveLayer.c
** These routines provide a layer of support between my bare-
bones application skeleton and the Bezier curve code.
There's a little here of interest outside of the mouse
tracking and the curve drawing.
** David W. Smith
*/

#include "QuickDraw.h"
#include "MacTypes.h"
#include "FontMgr.h"
#include "WindowMgr.h"
#include "MenuMgr.h"
#include "TextEdit.h"
#include "DialogMgr.h"
#include "EventMgr.h"
#include "DeskMgr.h"
#include "FileMgr.h"
#include "ToolboxUtil.h"
#include "ControlMgr.h"

/*
* Tracker objects. Similar to MacAPP trackers, but much,
much simpler.
*/
struct Tracker
{
    void (*track)();
    int  thePoint;
};

static struct Tracker aTracker;
static struct Tracker bTracker;

/*
* The Bezier curve control points.
*/
Point    control[4] = {(144,72), (72,144), (216,144),
(144,216)};

/*
* Draw
* Called from the skeleton to update the window. Draw the
initial curve.
*/
Draw()
{
    PenMode(patXor);
    DrawTheCurve(control, true);
}

```

```

/*
* DrawTheCurve
* Draw the given Bezier curve in the current pen mode. Draw
the control points if requested.
*/
DrawTheCurve(c, drawPoints)
    Point    c[];
{
    if ( drawPoints )
        DrawThePoints(c);
    BezierCurve(c[0], c[1], c[2], c[3]);
}

/*
* DrawThePoints
* Draw all of the control points.
*/
DrawThePoints(c)
    Point    c[];
{
    int      n;

    for ( n = 0 ; n < 4 ; ++n ) {
        DrawPoint(c, n);
    }
}

/*
* DrawPoint
* Draw a single control point
*/
DrawPoint(c, n)
    Point    c[];
    int      n;
{
    PenSize(3, 3);
    MoveTo(c[n].h - 1, c[n].v - 1);
    LineTo(c[n].h - 1, c[n].v - 1);
    PenSize(1, 1);
}

/*
* GetTracker
* Produce a tracker object
* Called by the skeleton to handle mouse-down events.
* If the mouse touches a control point, return a tracker for
that point. Otherwise, return a tracker that drags a gray
rectangle.
*/
struct Tracker *
GetTracker(point)
    Point    point;
{
    void      TrackPoint(), TrackSelect();
    int      i;

    aTracker.track = TrackPoint;

    for ( i = 0 ; i < 4 ; ++i ) {
        if ( TouchPoint(control[i], point) ) {
            aTracker.thePoint = i;
            return (&aTracker);
        }
    }
    bTracker.track = TrackSelect;
    return (&bTracker);
}

/*
* TouchPoint
* Do the points touch?
*/
#define abs(a) (a < 0 ? -(a) : (a))

```



```

TouchPoint(target, point)
    Point    target;
    Point    point;
    {
        SubPt(point, &target);
        if ( abs(target.h) < 3 && abs(target.v) < 3 )
            return (1);
        return (0);
    }

/*
 * TrackPoint
 * Called while dragging a control point.
 */
void
TrackPoint(tracker, point, phase)
    struct Tracker *tracker;
    Point    point;
    int      phase;
    {
        Point    savePoint;

        switch ( phase ) {
        case 1:
            /* initial click - XOR out the control point */
            DrawPoint(control, tracker->thePoint);
            break;
        case 2:
            /* drag - undraw the original curve and draw the new one */
            DrawTheCurve(control, false);
            control[tracker->thePoint] = point;
            DrawTheCurve(control, false);
            break;
        case 3:
            /* release - redraw the control point */
            DrawPoint(control, tracker->thePoint);
            break;
        }
    }

/*
 * TrackSelect
 * Track a gray selection rectangle
 */
static Point    first;
static Rect     r;

void
TrackSelect(tracker, point, phase)
    struct Tracker *tracker;
    Point    point;
    int      phase;
    {
        switch ( phase ) {
        case 1:
            PenPat(gray);
            first = point;
            SetupRect(&r, first, point);
            FrameRect(&r);
            break;
        case 2:
            FrameRect(&r);
            SetupRect(&r, first, point);
            FrameRect(&r);
            break;
        case 3:
            FrameRect(&r);
            PenPat(black);
            break;
        }
    }
}

```

```

 * SetupRect
 * Setup the rectangle for tracking.
 */
#define min(x, y) (((x) < (y)) ? (x) : (y))
#define max(x, y) (((x) > (y)) ? (x) : (y))

SetupRect(rect, point1, point2)
    Rect    *rect;
    Point    point1;
    Point    point2;
    {
        SetRect(rect,
            min(point1.h, point2.h),
            min(point1.v, point2.v),
            max(point1.h, point2.h),
            max(point1.v, point2.v));
    }

/*
 ** Skeleton.c - A bare-bones skeleton.
 ** This has been hacked up to demonstrate Bezier curves.
 ** Other than the tracking technique, there's little here of
 ** interest.
 ** David W. Smith
 */

#include "QuickDraw.h"
#include "MacTypes.h"
#include "FontMgr.h"
#include "WindowMgr.h"
#include "MenuMgr.h"
#include "TextEdit.h"
#include "DialogMgr.h"
#include "EventMgr.h"
#include "DeskMgr.h"
#include "FileMgr.h"
#include "ToolboxUtil.h"
#include "ControlMgr.h"

WindowRecord wRecord;
WindowPtr    myWindow;

/*
 * main
 * Initialize the world, then handle events until told to
 * quit.
 */
main()
    {
        InitGraf(&thePort);
        InitFonts();
        FlushEvents(everyEvent, 0);
        InitWindows();
        InitMenus();
        InitDialogs(0L);
        InitCursor();
        MaxApplZone();

        SetupMenus();
        SetupWindow();
        SetupBezier();

        while ( DoEvent(everyEvent) )
            ;
    }

/*
 * SetupMenus
 * For the purpose of this demo, we get somewhat non-standard
 * and use no menus. Closing the window quits.
 */
SetupMenus()
    {
        DrawMenuBar();
    }

```

```

}

/*
 * SetupWindow
 * Setup the window for the Bezier demo.
 */
SetupWindow()
{
    Rect    bounds;

    bounds = WMgrPort->portBits.bounds;
    bounds.top += 36;
    InsetRect(&bounds, 5, 5);

    myWindow = NewWindow(&wRecord, &bounds, "\pBezier Sampler -
Click and Drag", 1, noGrowDocProc, 0L, 1, 0L);

    SetPort(myWindow);
}

/*
 * DoEvent
 * Generic event handling.
 */
DoEvent(eventMask)
    int    eventMask;
{
    EventRecord    myEvent;
    WindowPtr    whichWindow;
    Rect    r;

    SystemTask();
    if ( GetNextEvent(eventMask, &myEvent) )
    {
        switch ( myEvent.what )
        {
            case mouseDown:
                switch ( FindWindow( myEvent.where, &whichWindow ) )
                {
                    case inDesk:
                        break;
                    case inGoAway:
                        if ( TrackGoAway(myWindow, myEvent.where) )
                        {
                            HideWindow(myWindow);
                            return (0);
                        }
                        break;
                    case inMenuBar:
                        return (DoCommand(MenuSelect(myEvent.where)));
                    case inSysWindow:
                        SystemClick(&myEvent, whichWindow);
                        break;
                    case inDrag:
                        break;
                    case inGrow:
                        break;
                    case inContent:
                        DoContent(&myEvent);
                        break;
                    default:
                        break;;
                }
                break;
            case keyDown:
            case autoKey:
                break;
            case activateEvt:
                break;
            case updateEvt:
                DoUpdate();
                break;
            default:
                break;
        }
    }
}

```

```

    }
    return(1);
}

/*
 * DoCommand
 * Command handling would normally go here.
 */
DoCommand(mResult)
    long    mResult;
{
    int    theItem, temp;
    Str255    name;
    WindowPeek    wPtr;

    theItem = LoWord(mResult);

    switch ( HiWord(mResult) )
    {
        {

        HiliteMenu(0);
        return(1);
    }

/*
 * DoUpdate
 * Generic update handler.
 */
DoUpdate()
{
    BeginUpdate(myWindow);
    Draw();
    EndUpdate(myWindow);
}

/*
 * DoContent
 * Handle mouse-downs in the content area by asking the
application to produce a tracker object. We then call the
tracker repeatedly to track the mouse. This technique came
originally (as nearly as I can tell) from Xerox, and is used
in a modified form in MacApp.
 */
struct Tracker
{
    int    (*Track)();
};

int
DoContent(pEvent)
    EventRecord    *pEvent;
{
    struct Tracker    *GetTracker();
    struct Tracker    *t;
    Point    point, newPoint;

    point = pEvent->where;
    GlobalToLocal(&point);
    t = GetTracker(point);
    if ( t ) {
        (*t->Track)(t, point, 1);
        while ( StillDown() ) {
            GetMouse(&newPoint);
            if ( newPoint.h != point.h || newPoint.v != point.v ) {
                point = newPoint;
                (*t->Track)(t, point, 2);
            }
        }
        (*t->Track)(t, point, 3);
    }
}

```



C Workshop

Event Simulator



Matthew J. Snyder
Fairfield, CA
Volume 5, Number 1

[Matthew Snyder has a B.S.E.E. and a B.S.(Mathematics) from the University of Notre Dame. Upon graduation, He was commissioned an Ensign in the U.S. Navy Civil Engineer Corps and is serving as a construction contract administrator in a field office at Mare Island Naval Shipyard in Vallejo, California (North Bay area).]

Introduction

There are several products available today (including the newest system software) which let you play back arbitrary sequences of keyboard and mouse events at the touch of a button, and which can save you lots of time and effort if you need to execute the same pattern over and over. The playback involves a sort of simulation in which it is made to appear to the active application that the user is producing the events with his own fingers.

What I provide in this article is a poor man's event simulator. It's a code library, not a macro maker. There is nothing tricky about it, and it doesn't violate any rules that I know of. The simulator makes it easy for the programmer to place simulated events onto the event queue, and the routines called by the event simulator are well-documented.

Background

The Operating System normally takes input from the user and stores events in the event queue, a standard Operating System queue. The Toolbox Event Manager in turn takes events from the event queue, as well as from other places, and passes them to the application in response to `GetNextEvent's`. My routines simply imitate the Operating System by creating event records and placing them on the event queue by hand.

The Demo

I wrote the library at a time when I had a need for an FKEY that would generate a few mousedowns. Having an FKEY that produces mousedowns is a little like having one of the commercial keyboard macro products I mentioned in the first paragraph, and in this article I show an example of such an FKEY. However, the resulting keyboard macro is not very elegant, it's limited, and it's difficult to install and use.

In fact, I am not really recommending frequent use of such an FKEY. I present the FKEY example merely to illustrate a possible use for the simulator. I hope that readers can think of more creative and less dangerous ways to use the simulator.

How the Simulator Works

I chose not to make use of the Toolbox Event Manager's journaling mechanism. A look at the documentation in Inside Macintosh will reveal that on one hand it is a complicated matter,

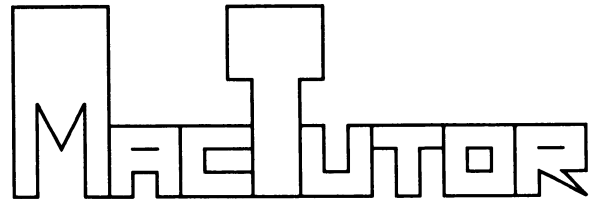


Figure 1. THE journal for the elite

involving the use of device drivers to both record and play back an event journal, and on the other hand it is poorly documented. I have yet to see a good example of its use.

Below, I present the C declaration of the data structure used to maintain the toolbox event queue. For the Lightspeed C system, the declaration is found in `<EventMgr.h>`:

```
typedef struct EvQE1
{
    struct QE1em *qLink;
    int    qType;
    int    evtQWhat;
    long   evtQMessage;
    long   evtQWhen;
    Point  evtQWhere;
    int    evtQModifiers;
} RealEvQE1, *RealEvQE1Ptr;
```

When I was implementing FKEY macros, I found it convenient to place a delay between some events. To do this, I install a routine into the VBL task queue that enqueues an event after a certain delay. To make it easier to install the routine, I wrote a little interface code and placed it in the simulator library. The routine is called `MakeTimed`, and the data structure needed for the VBL queue is declared this way in Lightspeed's `<VRetraceMgr.h>`:

```
typedef struct VBLTask
{
    struct QE1em *qLink;
    int    qType;
    ProcPtr vblAddr;
    int    vblCount;
    int    vblPhase;
} VBLTask, *VBLQE1Ptr;
```

Warnings

I discovered a few other things while making the FKEY macros, and a little thought about these things will indicate why the use of the simulator in an FKEY is not the greatest idea. An FKEY's code segment becomes locked in the heap while it is executing, but becomes unlocked after it completes. If the FKEY routine makes use of the `MakeTimed` function, and places the

address of a routine in its code segment onto the VBL task queue, it should lock its own segment down. Doing this after the FKEY completes is not a simple matter.

The simulator allocates dynamic memory to hold the data structures whose addresses get enqueued to the toolbox event queue, so locking the code segment is not usually necessary. If you're wondering what happens to the memory that gets enqueued to the event queue, I understand because I wondered too. I tracked one block, but didn't learn anything interesting. The memory never becomes deallocated, either by the simulator or the Operating System. Because it's allocated in the application heap, it returns to the pool when the application exits, but this may be little consolation to you if you want to generate a lot of events. A possible solution would be to add a VBL task that tracks the blocks and deallocates them when they've served their purpose. A tracker would add a lot of complexity to the simulator, and is left to the reader.

How the Demo Works

A few words about the demonstration FKEY and its algorithm: It's designed for use with pre-Claris MacDraw (I have not tested it with the most current version). Its only function is the automatic generation of a picture. It does so using a technique for drawing lines by generating a mousedown in one location followed by a mouseup in another location.

I have found that MacDraw is the only software stupid enough (or smart enough, I'm not sure which) to be fooled by this technique. If you think about it, a line-drawing application should probably be tracking the mouse after a mousedown, drawing and redrawing the line until the mouse button is released. MacDraw has no problem with the mouse action occurring too quickly to track.

The Menu Manager is much too smart to be fooled by the technique, so it's impossible to make an automatic menu selection unless the item has a command key equivalent.

A few words about the code itself: If you're wondering how I determined the global coordinates of each point in the drawing, I used Rick Flott's Mouse Position DA, published in MacTutor (I have The Complete MacTutor Vol. 2, so I don't know from which issue the article came). I used the DA in combination with Apple's MouseKeys.

More about the demo code: An FKEY runs in the context of another program, so it cannot have global data of its own. However, Lightspeed provides a facility for embedding data at the end of the code block, and accessing it as if it were global data. It's almost completely transparent to the programmer. You simply declare the variables as global, autoinitialize them as you would real global variables, and reference them like any other variable. Lightspeed uses register A4 to point to the data space, and translates any use of the variable into a reference off A4.

The only explicit action the programmer must make is to place the address of the beginning of the data space into A4 before using the variables. As I mentioned above, the data is placed at the end of the code block. But Lightspeed uses the BEGINNING of the code block as a base in its references. For example, if your code turns out to take up 100 bytes, the first pseudo-global would

be referenced as 100(A4).

So the address of the beginning of the data space is the same as the address of the beginning of the code block, which is the same as the dereferenced value of the handle that you would use to lock and unlock the block. This value is conveniently passed to the code resource when it is called, in register A0. I use the macro recommended in the Lightspeed manual for initializing and restoring register A4.

How to Make the Demo Work

To use the demonstration FKEY, build the project with type set to Code Resource. It's 4-character TYPE should be FKEY, and its ID should be 8 (or any number you prefer, between 0 and 9). Use ResEdit or its equivalent to place the FKEY resource of the resulting document into a backup copy of MacDraw. Placing the FKEY in the application itself is preferred to placing it in the System file, since accidentally using the FKEY elsewhere would be undesirable.

Start up MacDraw, and hit command-shift-8. The rest is automatic.

Name	obj size
MakeKey.o	132
MakeMouse.o	44
MakeMouseDown.o	160
MakeMouseUp.o	160
MakeTimed.o	82

Name	obj size
demo.o	322
EvSimulator.lib	578
MacTraps	6622

Figure 2. The Projects

```

/*****
** MakeKey.c **
*****/

#include <OSUtil.h>
#include <EventMgr.h>

MakeKey(Code, Mods)
int Code, Mods;
{

/* locals */

EvQE1Ptr MyEventPtr;
QHdPtr TheHdr;

/* begin executable */

/*1: Key Down */

MyEventPtr = (EvQE1Ptr) NewPtr ( sizeof(EvQE1) );

TheHdr = GetEvQHd();

MyEventPtr->qType = evType;
MyEventPtr->evtQWhat = 3;
MyEventPtr->evtQMessage = Code;
MyEventPtr->evtQWhen = TickCount();
MyEventPtr->evtQWhere.h = 200;
MyEventPtr->evtQWhere.v = 200;
MyEventPtr->evtQModifiers = Mods;

Enqueue (MyEventPtr, TheHdr);

/*2: Key Up */

```

```

/** For most applications, Key Up is unneeded */
MyEventPtr = (EvQE1Ptr) NewPtr ( sizeof(EvQE1) );

MyEventPtr->qType = evType;
MyEventPtr->evtQWhat = 4;
MyEventPtr->evtQMessage = Code;
MyEventPtr->evtQWhen = TickCount();
MyEventPtr->evtQWhere.h = 361;
MyEventPtr->evtQWhere.v = 98;
MyEventPtr->evtQModifiers = Mods;

Enqueue (MyEventPtr, TheHdr);

/** For most applications, Key Up is unneeded */

}

/*****
** MakeMouse.c **
*****/

MakeMouse(hzntl,vrtcl)
int hzntl,vrtcl;
{

MakeMouseDown(hzntl,vrtcl);
MakeMouseUp(hzntl,vrtcl);

}

/*****
** MakeMouseDown.c **
*****/

#include <OSUtil.h>
#include <EventMgr.h>

/* mouse down, no key modifier */

MakeMouseDown(hzntl,vrtcl)
int hzntl,vrtcl;
{

MMDmod(hzntl,vrtcl,0);

}

/* mouse down with key modifier */

MMDmod(hzntl,vrtcl,mod)
int hzntl,vrtcl,mod;
{

/* locals */

EvQE1Ptr MyEventPtr;
QHdrPtr TheHdr;

/* begin executable */

MyEventPtr = (EvQE1Ptr) NewPtr ( sizeof(EvQE1) );

TheHdr = GetEvQHdr();

MyEventPtr->qType = evType;
MyEventPtr->evtQWhat = 1;
MyEventPtr->evtQMessage = 0;
MyEventPtr->evtQWhen = TickCount();
MyEventPtr->evtQWhere.h = hzntl;
MyEventPtr->evtQWhere.v = vrtcl;
MyEventPtr->evtQModifiers = mod;

```

```

Enqueue (MyEventPtr, TheHdr);

}

/*****
** MakeMouseUp.c **
*****/

#include <OSUtil.h>
#include <EventMgr.h>

/* mouse up, no key modifier */

MakeMouseUp(hzntl,vrtcl)
int hzntl,vrtcl;
{

MMUmod(hzntl,vrtcl,0);

}

/* mouse up with key modifier */

MMUmod(hzntl,vrtcl,mod)
int hzntl,vrtcl,mod;
{

/* locals */

EvQE1Ptr MyEventPtr;
QHdrPtr TheHdr;

/* begin executable */

MyEventPtr = (EvQE1Ptr) NewPtr ( sizeof(EvQE1) );

TheHdr = GetEvQHdr();

MyEventPtr->qType = evType;
MyEventPtr->evtQWhat = 2;
MyEventPtr->evtQMessage = 0;
MyEventPtr->evtQWhen = TickCount();
MyEventPtr->evtQWhere.h = hzntl;
MyEventPtr->evtQWhere.v = vrtcl;
MyEventPtr->evtQModifiers = mod;

Enqueue (MyEventPtr, TheHdr);

}

/*****
** MakeTimed.c **
*****/

#include <OSUtil.h>
#include <VReTraceMgr.h>

MakeTimed(Func,Ticks)
ProcPtr Func;
int Ticks;
{

/* locals */

VBLQE1Ptr MyVBLptr;

/* begin executable */

MyVBLptr = ( VBLQE1Ptr ) NewPtr ( sizeof( VBLTask ) );

MyVBLptr->qType = vType;
MyVBLptr->vblAddr = Func;
MyVBLptr->vblCount = Ticks;
MyVBLptr->vblPhase = 0;

```

```

VInstall (MyVBLptr);
}

/*****
** demo.h **
*****/

/* struct for handling large sets of points */
typedef struct letter {
    int *h;
    int *v;
} letter;

/* functions used to get access to psuedo-globals */

#define SetUpA4() asm (    MOVE.L A4, -(SP)\
                        MOVE.L A0, A4 )

#define RestoreA4() asm (    MOVE.L (SP)+, A4 )

/*****
** demo.c **
*****/

#include "demo.h"

/* psuedo-global data points */

int data1 [20] = { 119, 119, 128, 128, 137, 146, 146, 155, 155, 119, 0
};
int data2 [20] = { 115, 187, 187, 151, 169, 151, 187, 187, 115, 115, 0
};

int data3 [20] = { 155, 155, 164, 164, 173, 173, 182, 182, 155, 0,
164, 173, 0 };
int data4 [20] = { 160, 187, 187, 178, 178, 187, 187, 160, 160, 0,
169, 169, 0 };

int data5 [20] = { 182, 182, 209, 209, 191, 191, 209, 209, 182, 0 };
int data6 [20] = { 160, 187, 187, 178, 178, 169, 169, 160, 160, 0 };

int data7 [20] = { 200, 200, 209, 209, 227, 227, 236, 236, 200, 0 };
int data8 [20] = { 115, 142, 142, 187, 187, 142, 142, 115, 115, 0 };

int data9 [20] = { 227, 227, 254, 254, 245, 245, 236, 236, 227, 0 };
int data10 [20] = { 160, 187, 187, 160, 160, 178, 178, 160, 160, 0 };

int data11 [20] = { 254, 254, 263, 263, 272, 272, 281, 281, 254, 0 };
int data12 [20] = { 160, 169, 169, 187, 187, 169, 169, 160, 160, 0 };

int data13 [20] = { 281, 281, 308, 308, 281, 0,
290, 290, 299, 299, 290, 0 };
int data14 [20] = { 160, 187, 187, 160, 160, 0,
169, 178, 178, 169, 169, 0 };

int data15 [20] = { 308, 308, 317, 317, 335, 326, 335, 335, 308, 0,
317, 326, 0 };
int data16 [20] = { 160, 187, 187, 179, 187, 178, 178, 160, 160, 0,
169, 169, 0 };

/* main routine */

main() {

/* locals */

letter M, a, c, T, u, t, o, r;
int h_index, v_index;

/* begin executable code */

```

```

/* access psuedo-globals */
SetUpA4();

/* load structures */

M.h = data1;
a.h = data3;
c.h = data5;
T.h = data7;
u.h = data9;
t.h = data11;
o.h = data13;
r.h = data15;

M.v = data2;
a.v = data4;
c.v = data6;
T.v = data8;
u.v = data10;
t.v = data12;
o.v = data14;
r.v = data16;

/* select from palette */
MakeMouse(15, 100);

/* draw letters */
DrawLetter(M.h, M.v);
DrawLetter(a.h, a.v); DrawLetter(&a.h[10], &a.v[10]);
DrawLetter(c.h, c.v);
DrawLetter(T.h, T.v);
DrawLetter(u.h, u.v);
DrawLetter(t.h, t.v);
DrawLetter(o.h, o.v); DrawLetter(&o.h[6], &o.v[6]);
DrawLetter(r.h, r.v); DrawLetter(&r.h[10], &r.v[10]);

/* deselect last line */
MakeMouse(400, 250);

/* pop A4 */
RestoreA4();

}

/* routine to draw letters */

DrawLetter ( hArray, vArray )
int hArray[], vArray[];
{

int h_index, v_index;

for (h_index=0, v_index=0; (hArray[h_index+1] != 0) ||
(vArray[v_index+1] != 0); h_index++, v_index++) {
    MMUmod(hArray[h_index], vArray[v_index], 0x0100);
    MMUmod(hArray[h_index+1], vArray[v_index+1], 0x0100);
}

}

```



C Workshop

How to Write a Spreadsheet in LS C



Bryan Waters
Casselberry, FL
Volume 5, Number 3

Bryan Waters is a Software Engineer for Maynard Electronics, and is currently working as part of a team to develop a finder-like tape backup system.

Dear Cary Mariash

In the January '88 issue of MacTutor, a letter sent by Cary Mariash requested information on Macintosh tools to create spreadsheet type windows. MacTutorCalc is an example of a simple (very simple!) spreadsheet type application, that uses the List Manager (IM - IV) to create it's windows. (Note: MacTutorCalc is written in Think C 3.0, see the project window in figure 1)

List Manager In action!

The call to Lnew is used to create the list (figure 2). The dataBounds parameter is set to have an extra row and column than our work area. This, of course, is to support the row and column headers. After the list has been created the programmer has several more options, involving selection, and installing a routine in the IClickLoop field to be called repeatedly while LClick has control. The selFlags field of the list handle allow complete customization of the selection algorithm used by the list manager. MacTutorCalc is content to use the default selection, but when we call LClick to handle our mouseDown events in the window, any scrolling causes the spreadsheet to scroll, without the grid being updated. To fix this, MacTutorCalc installs a IClickLoop routine to update the grid continuously during calls to LClick. This works great, except the List Manager does not call this routine when the mouse button is pressed in the scroll bars, so that the grid is updated then only after the mouse button is released. Although it is not implemented here, this could be fixed by installing a pointer to our grid routine in the system global DragHook (\$9F6) which is called by DragGrayRgn. Since the control manager routine TrackControl calls DragGrayRgn to drag the outline of the scroll bars thumb button, this would achieve the desired effect.

There are other points to take into account when using a window such as this. For example, when the user resizes the window, it would not be desirable to allow the user to size the window out of alignment with the cell boundaries simply because it looks bad. To handle this, after the call to GrowWindow, we make sure that the new size is a multiple of the cell size. Updating is handled almost completely by the list manager routine LUpdate, although we still have to draw the grid, and the grow icon.

Data entry is probably where MacTutorCalc could be improved the most. Currently data entry is implemented by double-clicking on the desired cell (LClick returns TRUE when a cell was double-clicked and a call to LLastClick is used to get the cell's address). This brings up a dialog prompting for the data.

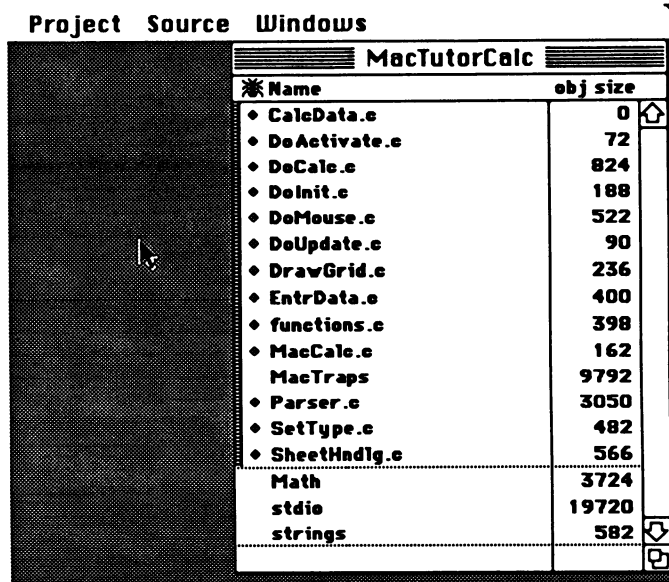


Figure 1. Lightspeed C Project Window

In an application intended for serious use, this would get extremely tiring. This could be fixed by adding a data entry window at the top of the screen (Excel Style) and allowing the user to enter data into the currently selected cell through this window. At this point we have data that needs to be processed, so we determine the type of data (formula, or string) and then set our calculation flag. The routine DoCalc is called in response to this, and processes the whole spreadsheet, calling the parser for formula,

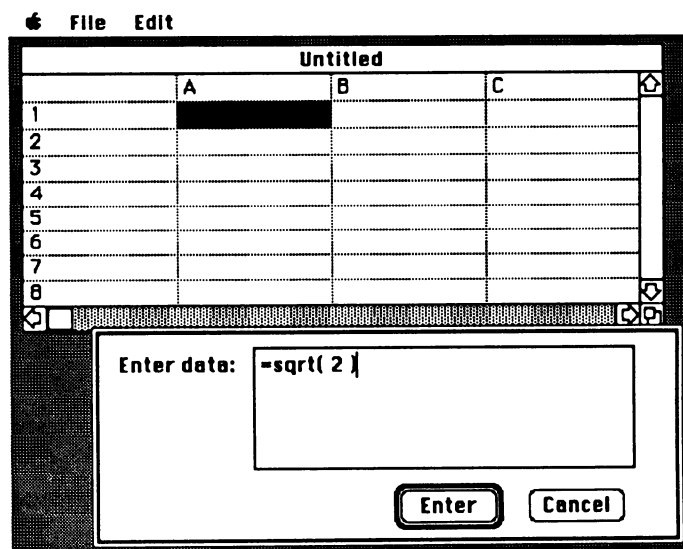


Figure 2. Spreadsheet & Dialog Entry

and ignoring strings. After the data has been parsed, LSetCell is used to put the data into the list. Alternatively, if the cell has been blanked out, then LClrCell is used to clear the data.

About MacTutorCalc...

MacTutorCalc is a simple spreadsheet written for the purpose of demonstrating use of the list manager. The work area for our application is set to 8 columns by 8 rows. It is capable of accepting both strings and formulae, and constants. Data is entered into a cell by double-clicking on it. I know that this is awkward, but other methods would have unnecessarily complicated this example. Any data entered into a cell that starts with "=" is considered a formula and will be parsed, and any number will be treated as a constant, everything else is considered a string. A formula can be any simple algebraic expression, using floating or absolute cell addresses. There are a few simple functions in MacTutorCalc that use the Math library distributed with Think C. In the future, we could expand our spreadsheet to include:

- ability to save and retrieve spreadsheets
- a full function library
- cell formatting and spreadsheet editing features
- memory paging system to support larger spreadsheet sizes
- support for SYLK format

[There is a known bug in the text to binary conversion routines. Apple's conversion routines were used, and so large formatted numbers will not be converted to text correctly. Note also that Apple frowns on trying to use the List Manager to write the next Excel Product, so if you get serious about this project, re-write it without using the List Manager. -Ed]

Listing: CalcData.h

```
/* Global data */
extern int quit_flag ;

extern Rect minmax_size ;
extern Rect curr_screen ;
extern int automatic_calculation ;
extern int calc_data ;
extern int do_calc_now ;
extern SHEET_WIN_PTR curr_sheet_ptr ;
extern SHEET_WIN_HDL calc_hdl ;
extern FUN_ENTRY fun_table[] ;
extern ARG arg_free_pool[] ;
```

Listing: MacCalc.h

```
#include <MacTypes.h>
#include <ListMgr.h>
#ifdef NULL
#define NULL 0L
#endif
#define MAX_ROWS 8
#define MAX_COLUMNS 8
#define CELL_WIDTH 96
#define CELL_HEIGHT 16

#define ENTER_DATA_DIALOG 128
#define APPLE_MENU 1
#define FILE_MENU 256
#define EDIT_MENU 257

/* Types for cell */
#define CLEARED -1
#define UNDEFINED 0
```

```
#define STRING 1
#define FORMULA 2
#define CONSTANT 3

struct cell_struct(
    int type ;
    double value ;
    Str255 formula ;
) ;

typedef struct cell_struct CELL, *CELL_PTR, **CELL_HDL ;

struct sheet_win(
    WindowPtr sheet_window_ptr ;
    ListHandle sheet_list_hdl ;
    CELL sheet_data[MAX_ROWS][MAX_COLUMNS] ;
) ;

typedef struct sheet_win SHEET_WIN, *SHEET_WIN_PTR,
**SHEET_WIN_HDL ;

/* Argument types */
#define VALUE_ARG 1
#define STRING_ARG 2
/* Arg usage defines */
#define IN_USE TRUE
#define FREE_ARG FALSE

struct fun_arg(
    struct fun_arg *next_arg ;
    int in_use ;
    int type ;
    double value ;
    Str255 string ;
) ;

typedef struct fun_arg ARG, *ARG_PTR, **ARG_HDL ;
struct fun_entry(
    char fun_name[32] ;
    double (*fun_ptr)() ;
) ;

typedef struct fun_entry FUN_ENTRY, *FUN_ENTRY_PTR,
**FUN_ENTRY_HDL ;

/* Prototypes */
int DoInit( void ) ;
void DoEventLoop( void ) ;
void DoActivate( EventRecord * ) ;
void DoUpdate( EventRecord * ) ;
void DoMouseDown( EventRecord * ) ;

int ClickLoop( void ) ;
void EnterData( Cell, SHEET_WIN_HDL ) ;
void DoCalc( SHEET_WIN_HDL ) ;
```

Listing: SheetHndl.h

```
/* Spreadsheet Handling prototypes */
int OpenNewSpreadsheet( char * ) ;
int DrawGrid( WindowPtr ) ;
```

Listing: Parser.h

```
/* Parse errors */
#define MISMATCHED_PARENTHESIS 200
#define INVALID_NUMBER 201
#define INVALID_ADDRESS 202
#define ADDRESS_TOO_LARGE 203
#define INVALID_FUNCTION 204

/* Prototypes */
void SetType( CELL_PTR, int ) ;
double ParseFormula( unsigned char *, int * ) ;
```


© The Best of MacTutor, Vol. 5

```

/* If cell type is undefined then do nothing */
case UNDEFINED:
    break ;
/* If cell type is undefined then clear cell */
case CLEARED:
    SetPt( &curr_cell, j+1, i+1 ) ;
    LClrCell( curr_cell, curr_sheet_ptr-
>sheet_list_hdl ) ;
    curr_sheet_ptr->sheet_data[i][j].type =
UNDEFINED ;
    break ;
/* If cell type is string, put string in cell */
case STRING:
    SetPt( &curr_cell, j+1, i+1 ) ;
    LSetCell( &curr_sheet_ptr-
>sheet_data[i][j].formula[i],
    curr_sheet_ptr-
>sheet_data[i][j].formula[0],
    curr_cell,
    curr_sheet_ptr->sheet_list_hdl ) ;
    break ;
case CONSTANT:
    SetPt( &curr_cell, j+1, i+1 ) ;
    curr_sheet_ptr->sheet_data[i][j].value =
    GetFloat( curr_sheet_ptr-
>sheet_data[i][j].formula, &error ) ;
    if( error ) {
        Alert( error, NULL ) ;
        err_mess = "\p***ERROR***" ;
        LSetCell( err_mess+1, *err_mess,
curr_cell, curr_sheet_ptr->sheet_list_hdl ) ;
    } else {
        ftoa( curr_sheet_ptr-
>sheet_data[i][j].value, buffer ) ;
        LSetCell( &buffer[1],
            buffer[0],
            curr_cell,
            curr_sheet_ptr->sheet_list_hdl ) ;
    }
    break ;
/* Parse formula then convert value to string and
put in cell */
case FORMULA:
    SetPt( &curr_cell, j+1, i+1 ) ;
    curr_sheet_ptr->sheet_data[i][j].value =
    ParseFormula( curr_sheet_ptr-
>sheet_data[i][j].formula, &error ) ;
    if( error ) {
        Alert( error, NULL ) ;
        err_mess = "\p***ERROR***" ;
        LSetCell( err_mess+1, *err_mess,
curr_cell, curr_sheet_ptr->sheet_list_hdl ) ;
    } else {
        ftoa( curr_sheet_ptr-
>sheet_data[i][j].value, buffer ) ;
        LSetCell( &buffer[1],
            buffer[0],
            curr_cell,
            curr_sheet_ptr->sheet_list_hdl ) ;
    }
    break ;
}
}
}
calc_data = FALSE ;
DrawGrid( curr_sheet_ptr->sheet_window_ptr ) ;
HUnlock( (Handle)sheet_record_hdl ) ;
return ;
}

```

Listing: DoInit.c

```

#include <WindowMgr.h>
#include <ListMgr.h>

```

```

#include <OSUtil.h>
#include <EventMgr.h>

#include "MacCalc.h"
#include "SheetHndlg.h"
#include "CalcData.h"

void CreateMenus( void ) ;

int DoInit( )
{
    /* initialize all the Macintosh Managers */
    InitGraf( (Ptr)&thePort ) ; /* initialize quickdraw */
    InitFonts( ) ; /* initialize the font manager */
    InitWindows( ) ; /* initialize the window manager */
    InitCursor( ) ; /* initialize the cursor */
    InitMenus( ) ; /* initialize the menu manager */
    TEInit( ) ; /* initialize the text edit manager */
    InitDialogs( NULL ) ; /* initialize dialog manager */

    /* Flush events */
    FlushEvents( everyEvent, 0 ) ;

    /* Get curr screen size */
    curr_screen = screenBits.bounds ;
    SetRect( &minmax_size, 207, 77, 447, 159 ) ;

    /* Expand heap to maximum and allocate more masters */
    MaxApplZone( ) ;
    MoreMasters( ) ;
    MoreMasters( ) ;
    MoreMasters( ) ;
    MoreMasters( ) ;
    MoreMasters( ) ;

    CreateMenus( ) ;

    OpenNewSpreadsheet( "\pUntitled" ) ;
}
void CreateMenus( )
{
    MenuHandle menu ;

    menu = GetMenu( 1 ) ;
    AddResMenu( menu, 'DRVR' ) ;
    InsertMenu( menu, 0 ) ;
    InsertMenu( GetMenu( 256 ), 0 ) ;
    InsertMenu( GetMenu( 257 ), 0 ) ;
    DrawMenuBar( ) ;
}

```

Listing: DoMouse.c

```

#include <WindowMgr.h>
#include <ListMgr.h>
#include <OSUtil.h>
#include <EventMgr.h>

#include "MacCalc.h"
#include "SheetHndlg.h"
#include "CalcData.h"

WindowPtr curr_window_ptr ;

void DoMouseDown( ev )
EventRecord *ev ;
{
    SHEET_WIN_HDL sheet_record_hdl ;
    Point new_point ;
    int location ;
    GrafPtr tmp_port ;
    Rect boundsRect ;

```

```

long newsize ;
long menuchoice ;
int width, height ;
union {
    long tmp ;
    Cell last_cell ;
} a ;

location = FindWindow( ev->where, &curr_window_ptr ) ;

if( ( curr_window_ptr != FrontWindow( ) ) && (
curr_window_ptr != NULL ) ) {
    SelectWindow( curr_window_ptr ) ;
    return ;
}

new_point = ev->where ;
GetPort( &tmp_port ) ;
SetPort( curr_window_ptr ) ;
GlobalToLocal( &new_point ) ;

switch( location ) {
case inDesk:
    break ;
case inMenuBar:
    menuchoice = MenuSelect( ev->where ) ;
    if( HiWord( menuchoice ) == FILE_MENU ) {
        quit_flag = TRUE ;
    }
    HiliteMenu( 0 ) ;
    break ;
case inContent:
    sheet_record_hdl =
(SHEET_WIN_HDL)GetWRefCon( curr_window_ptr ) ;
    if( LClick( new_point, ev->modifiers,
(**sheet_record_hdl).sheet_list_hdl ) ) {
        a.tmp = LLastClick((**sheet_record_hdl).sheet_list_hdl ) ;
        EnterData( a.last_cell, sheet_record_hdl ) ;
    }
    DrawGrid( curr_window_ptr ) ;
    break ;
case inDrag:
    boundsRect = curr_screen ;
    DragWindow( curr_window_ptr, ev->where, &boundsRect ) ;
    break ;
case inGrow:
    sheet_record_hdl = (SHEET_WIN_HDL)GetWRefCon(
curr_window_ptr ) ;
    newsize = GrowWindow( curr_window_ptr, ev->where,
&minmax_size ) ;
    if( newsize ) {
        width = ( ( LoWord( newsize ) / CELL_WIDTH ) *
CELL_WIDTH ) + 15 ;
        height = ( ( HiWord( newsize ) / CELL_HEIGHT ) *
CELL_HEIGHT ) + 15 ;
        SizeWindow( curr_window_ptr, width, height, TRUE ) ;
        LSize( width-15, height-15,
(**sheet_record_hdl).sheet_list_hdl ) ;
        DrawGrid( curr_window_ptr ) ;
        DrawGrowIcon( curr_window_ptr ) ;
    }
    break ;
case inGoAway:
    if( TrackGoAway( curr_window_ptr, ev->where ) ) {
        quit_flag = TRUE ;
    }
    break ;
}
SetPort( tmp_port ) ;

return ;
}
int ClickLoop( )

```

```

(
    DrawGrid( curr_window_ptr ) ;
    return TRUE ;
)

```

Listing: DoUpdate.c

```

#include <WindowMgr.h>
#include <ListMgr.h>
#include <OSUtil.h>
#include <EventMgr.h>

#include "MacCalc.h"
#include "SheetHndlg.h"
#include "CalcData.h"

void DoUpdate( ev )
EventRecord *ev ;
{
    WindowPtr curr_window_ptr ;
    SHEET_WIN_HDL sheet_record_hdl ;

    curr_window_ptr = (WindowPtr)ev->message ;

    sheet_record_hdl = (SHEET_WIN_HDL)GetWRefCon(
curr_window_ptr ) ;

    BeginUpdate( curr_window_ptr ) ;

    LUpdate( curr_window_ptr->visRgn,
(**sheet_record_hdl).sheet_list_hdl ) ;

    DrawGrid( curr_window_ptr ) ;

    DrawGrowIcon( curr_window_ptr ) ;

    EndUpdate( curr_window_ptr ) ;

    return ;
}

```

Listing: DrawGrid.c

```

#include <ListMgr.h>
#include <WindowMgr.h>
#include <MacTypes.h>
#include <QuickDraw.h>

#include "MacCalc.h"
#include "SheetHndlg.h"

int DrawGrid( win_ptr )
WindowPtr win_ptr ;
{
    GrafPtr tmp_port ;
    Rect win_rect ;
    int bottom ;
    int right ;
    int num_h_lines ;
    int num_v_lines ;
    int i, max ;

    GetPort( &tmp_port ) ;
    SetPort( win_ptr ) ;

    PenNormal( ) ;
    PenPat( gray ) ;

    win_rect = win_ptr->portRect ;
    bottom = win_rect.bottom - 15 ;
    right = win_rect.right - 15 ;
    num_h_lines = bottom / CELL_HEIGHT ;
    num_v_lines = right / CELL_WIDTH ;

```

```

max = ( num_h_lines > num_v_lines ) ?
num_h_lines:num_v_lines ;

for( i = 1 ; i <= max ; i++ ) (
    if( i < num_h_lines ) (
        MoveTo( 0, i*CELL_HEIGHT ) ;
        LineTo( right, i*CELL_HEIGHT ) ;
    )
    if( i < num_v_lines ) (
        MoveTo( i*CELL_WIDTH, 0 ) ;
        LineTo( i*CELL_WIDTH, bottom ) ;
    )
)
PenNormal( ) ;
SetPort( tmp_port ) ;
return ;
}

```

Listing: EntrData.c

```

#include <ListMgr.h>
#include <DialogMgr.h>
#include <MemoryMgr.h>
#include <QuickDraw.h>

#include "MacCalc.h"
#include "SheetHndl.h"
#include "CalcData.h"
#include "Parser.h"

#define ENTER_BUTTON 1
#define CANCEL_BUTTON 2
#define DATA_ITEM 4

void EnterData( cell, sheet_record_hdl )
Cell cell ;
SHEET_WIN_HDL sheet_record_hdl ;
(
    DialogPtr new_dialog ;
    int item_hit ;
    Handle hdl ;
    Rect box ;
    int type ;
    int had_data = FALSE ;

    if( ( cell.v ) != 0 && ( cell.h != 0 ) ) (

        HLock( (Handle)sheet_record_hdl ) ;
        new_dialog = GetNewDialog( ENTER_DATA_DIALOG, NULL,
        (WindowPtr)-1L ) ;

        /* Outline button */
        SetPort( new_dialog ) ;
        GetItem( new_dialog, ENTER_BUTTON, &type, &hdl,&box)
;
        PenNormal( ) ;
        PenSize( 3,3 ) ;
        InsetRect( &box, -4,-4 ) ;
        FrameRoundRect( &box, 16,16 ) ;

        GetItem( new_dialog, DATA_ITEM, &type, &hdl, &box ) ;

        if( (**sheet_record_hdl).sheet_data[cell.v-1][cell.h-1].formula[0] != 0 ) (
            SetIText( hdl,
            (**sheet_record_hdl).sheet_data[cell.v-1][cell.h-1].formula ) ;
            SetIText( new_dialog, DATA_ITEM, 0, 255 ) ;
            had_data = TRUE ;
        )

        do(
            ModalDialog( NULL, &item_hit ) ;
        )while( item_hit == DATA_ITEM ) ;
    )

```

```

if( item_hit != CANCEL_BUTTON ) (
    GetIText( hdl,
    &(**sheet_record_hdl).sheet_data[cell.v-1][cell.h-1].formula);
    SetType( &(**sheet_record_hdl).sheet_data[cell.v-1][cell.h-1], had_data ) ;
    calc_hdl = sheet_record_hdl ;
    calc_data = TRUE ;
)
CloseDialog( new_dialog ) ;
HUnlock( (Handle)sheet_record_hdl ) ;
}else(
    SysBeep( 1 ) ;
)
return ;
}

```

Listing: functions.c

```

#include <WindowMgr.h>
#include <ListMgr.h>
#include <OSUtil.h>
#include <EventMgr.h>

#include <Math.h>

#include "MacCalc.h"
#include "CalcData.h"
#include "Parser.h"

double fsum( arg )
ARG_PTR arg ;
(
    double value = 0 ;

    value += arg->value ;

    while( arg->next_arg != NULL ) (
        arg = arg->next_arg ;
        value += arg->value ;
    )
    return value ;
)

double fabsolute( arg )
ARG_PTR arg ;
(
    register double value ;

    value = arg->value ;
    return( ( value != 0 ) ? value:-value ) ;
)

double fmodulus( arg )
ARG_PTR arg ;
(
    register double value1 ;
    register double value2 ;
    ARG_PTR curr_arg = arg ;

    value1 = arg->value ;
    value2 = arg->next_arg->value ;
    return( fmod( value1, value2 ) ) ;
)

double fsqrt( arg )
ARG_PTR arg ;
(
    register double value ;

    value = arg->value ;
    return( sqrt( value ) ) ;
)

```

Listing: MacCalc.c

```

#include <WindowMgr.h>
#include <ListMgr.h>

```

```

#include <OSUtil.h>
#include <EventMgr.h>

#include "MacCalc.h"
#include "SheetHndl.h"
#include "CalcData.h"
#include "parser.h"

main( )
{
    DoInit( ) ;
    DoEventLoop( ) ;
}

void DoEventLoop( )
{
    EventRecord ev ;

    while( !quit_flag ) {
        SystemTask( ) ;
        if( GetNextEvent( everyEvent, &ev ) ) {
            switch( ev.what ) {
                case mouseDown:
                    DoMouseDown( &ev ) ;
                    break ;
                case mouseUp:
                    break ;
                case keyDown:
                    break ;
                case keyUp:
                    break ;
                case activateEvt:
                    DoActivate( &ev ) ;
                    break ;
                case updateEvt:
                    DoUpdate( &ev ) ;
                    break ;
            }
        } else {
            if( ( automatic_calculation || do_calc_now ) &&
                calc_data ) {
                DoCalc( calc_hdl ) ;
                do_calc_now = FALSE ;
                calc_data = FALSE ;
            }
        }
    }
    return ;
}

```

Listing: Parser.c

```

#include <WindowMgr.h>
#include <ListMgr.h>
#include <OSUtil.h>
#include <EventMgr.h>

#include <Strings.h>
#include <Math.h>
#include <Ctype.h>
#include <Stdio.h>

#include "MacCalc.h"
#include "CalcData.h"
#include "Parser.h"

unsigned char *curr_pos = NULL ;
int parse_err = 0 ;
Str255 buffer ;

/* Simple algebraic expression parser */

double ParseFormula( formula, error )
unsigned char *formula ;

```

```

int *error ;
{
    register double value ;

    /* convert into c string in buffer */
    BlockMove( &formula[2], buffer, formula[0]-1 ) ;
    buffer[formula[0]-1] = 0 ; /* 0 terminate buffer */

    curr_pos = buffer ;
    parse_err = 0 ;

    v ( = ParseExpression( ) ;

    *error = parse_err ; /* Set error flag */

    return ( value ) ;
}

double ParseExpression( )
{
    register double val ;

    val = ParseFactor( ) ;

    switch( *curr_pos ) {
        case '+':
            curr_pos++ ;
            val += ParseExpression( ) ;
            break ;
        case '-':
            curr_pos++ ;
            val -= ParseExpression( ) ;
            break ;
        case '*':
            curr_pos++ ;
            val *= ParseExpression( ) ;
            break ;
        case '/':
            curr_pos++ ;
            val /= ParseExpression( ) ;
            break ;
    }
    return val ;
}

double ParseFactor( )
{
    register double val = 0 ;
    register double val2 ;
    register int has_sign = FALSE ;

    if( *curr_pos == '-' ) {
        has_sign = TRUE ;
        curr_pos++ ;
    }

    if( *curr_pos != '(' ) {
        /* determine whether this is an address or not */
        if( IsDigit( *curr_pos ) || *curr_pos == '.' ) {
            val = ParseValue( ) ;
        } else {
            if( IsFunction( ) ) {
                val = CallFunction( ) ;
            } else {
                val = ParseAddress( ) ;
            }
        }
    }

    switch( *curr_pos ) {
        case '^':
            curr_pos++ ;
            val2 = ParseFactor( ) ;
            val = pow( val, val2 ) ;
            break ;
        case '*':
            curr_pos++ ;

```

```

        val2 = ParseFactor( );
        val *= val2 ;
        break ;
    case '/':
        curr_pos++ ;
        val2 = ParseFactor( );
        val /= val2 ;
        break ;
    }
}
}else(
    curr_pos++ ;
    val = ParseExpression( ) ;
    if( *curr_pos == ')' ) (
        curr_pos++ ;
    )else(
        parse_err = MISMATCHED_PARENTHESES ;
    )
}
return ( has_sign ? -( val ):val ) ;
}
double ParseValue( )
(
    register double val = 0 ;

    if( IsDigit( *curr_pos ) || ( *curr_pos == '.' ) ) (
        val = atof( ) ;
    )else(
        parse_err = INVALID_NUMBER ;
    )
    return val ;
}
double ParseAddress( )
(
    register int row ;
    register int column ;
    register double val = 0 ;

    column = GetColumn( ) ;
    row = GetRow( ) ;

    if( ( row > MAX_ROWS ) || ( column > MAX_COLUMNS ) ) (
        parse_err = ADDRESS_TOO_LARGE ;
    )else(
        val = curr_sheet_ptr->sheet_data[row-1][column-
1].value ;
    )
    return val ;
}
int GetRow( )
(
    register int row ;

    if( IsDigit( *curr_pos ) ) (
        row = *curr_pos - '0' ;
    )else(
        parse_err = INVALID_ADDRESS ;
    )
    curr_pos++ ;

    return row ;
}
int GetColumn( )
(
    register int column ;

    if( IsAlpha( *curr_pos ) ) (
        if( *curr_pos >= 'a' ) (
            column = ( *curr_pos - 'a' ) + 1 ;
        )else(
            column = ( *curr_pos - 'A' ) + 1 ;
        )
    )else(
        parse_err = INVALID_ADDRESS ;

```

```

    )
    curr_pos++ ;
    return column ;
}
int IsFunction( )
(
    register unsigned char *p ;

    p = curr_pos ;
    while( IsAlpha( *p ) || IsDigit( *p ) ) (
        p++ ;
    )
    return ( *p == '(' ) ? TRUE:FALSE ;
)
double CallFunction( )
(
    unsigned char fun_name[32] ;
    register unsigned char *p ;
    register int fun_number ;
    register double value ;
    register ARG_PTR args ;

    p = fun_name ;
    while( IsAlpha( *curr_pos ) || IsDigit( *curr_pos ) ) (
        *p = toupper( *curr_pos ) ;
        p++ ; curr_pos++ ;
    )
    *p = '\0' ;
    curr_pos++ ;
    args = BuildArg( ) ;
    if( ( fun_number = Lookup( fun_name ) ) > -1 ) (
        value = (*fun_table[fun_number].fun_ptr)( args ) ;
    )else(
        parse_err = INVALID_FUNCTION ;
    )
    DestroyArgs( args ) ;
    return value ;
}
int Lookup( fun_name )
unsigned char *fun_name ;
(
    register int i = 0 ;

    while( fun_table[i].fun_name[0] != 0 ) (
        if( !strcmp( fun_name, fun_table[i].fun_name ) ) (
            return i ;
        )
        i++ ;
    )
    return -1 ;
}
ARG_PTR BuildArg( )
(
    int error ;
    ARG_PTR first_arg_ptr = NULL, last_arg_ptr = NULL,
    next_arg_ptr = NULL ;

    /* This needs to be changed so that recursive functions
    calls can be made */
    while( *curr_pos != ')' ) (
        next_arg_ptr = GetArg( ) ;
        next_arg_ptr->value = ParseExpression( ) ;
        next_arg_ptr->type = VALUE_ARG ;
        if( *curr_pos == ',' ) (
            curr_pos++ ;
        )
        if( last_arg_ptr != NULL ) (
            last_arg_ptr->next_arg = next_arg_ptr ;
        )
        if( first_arg_ptr == NULL ) (
            first_arg_ptr = next_arg_ptr ;
        )
        last_arg_ptr = next_arg_ptr ;
    )

```

```

next_arg_ptr->next_arg = NULL ;

/* Now pass the last parenthesis by */
if( *curr_pos == ')' ) {
    curr_pos++ ;
    return first_arg_ptr ;
}else{
    parse_err = MISMATCHED_PARENTHESIS ;
    return NULL ;
}
}

double atof( )
{
    register double val = 0.0 ;
    register double val2 = 0.0 ;

    if( IsDigit( *curr_pos ) || ( *curr_pos == '.' ) ) {
        while( *curr_pos && IsDigit( *curr_pos ) ) {
            val = ( val * 10 ) + ( *curr_pos - '0' ) ;
            curr_pos++ ;
        }
        if( *curr_pos == '.' ) {
            curr_pos++ ;
        }
        while( *curr_pos && IsDigit( *curr_pos ) ) {
            val2 = ( val2 * 10 ) + ( *curr_pos - '0' ) ;
            curr_pos++ ;
        }
        if( val2 != 0.0 ) {
            while( val2 > 1 ) {
                val2 /= 10 ;
            }
        }
        val += val2 ;
    }else{
        parse_err = INVALID_NUMBER ;
    }
    return val ;
}

void ftoa( val, buffer )
double val ;
unsigned char *buffer ;
{
    long whole ;
    Str255 buff ;
    Str255 tmp ;
    int has_sign ;
    int prec = /*curr_precision*/5 ;

    has_sign = ( val < 0 ) ? TRUE:FALSE ;
    whole = val/1 ;
    NumToString( whole, buffer ) ;
    /* convert into c string in buffer */
    BlockMove( &buffer[1], buff, buffer[0] ) ;
    buff[buffer[0]] = 0 ; /* 0 terminate buffer */
    val -= whole ;
    if( prec ) {
        do{
            val *= 10.0 ;
        }while( -prec ) ;
        whole = val ;
        NumToString( whole, buffer ) ;
        /* convert into c string in buffer */
        BlockMove( &buffer[1], tmp, buffer[0] ) ;
        tmp[buffer[0]] = 0 ; /* 0 terminate buffer */
        strcat( buff, "." ) ;
        strcat( buff, tmp ) ;
    }
    if( has_sign ) {
        tmp[0] = '-' ;
        tmp[1] = '\0' ;
        strcat( tmp, buff ) ;
        strcpy( &buffer[1], tmp ) ;
    }
}

```

```

        buffer[0] = strlen( tmp ) ;
    }else{
        strcpy( &buffer[1], buff ) ;
        buffer[0] = strlen( buff ) ;
    }

    return ;
}

double GetFloat( formula, error )
unsigned char *formula ;
int *error ;
{
    register double value ;

    /* convert into c string in buffer */
    BlockMove( &formula[1], buffer, formula[0] ) ;
    buffer[formula[0]] = 0 ; /* 0 terminate buffer */

    curr_pos = buffer ;
    parse_err = 0 ;
    value = atof( ) ;
    *error = parse_err ; /* Set error flag */
    return ( value ) ;
}

ARG_PTR GetArg( )
{
    register int i ;

    for( i=0;i<30;i++){
        if( arg_free_pool[i].in_use == FREE_ARG ) {
            arg_free_pool[i].in_use = IN_USE ;
            return &arg_free_pool[i] ;
        }
    }
    return NULL ;
}

void DestroyArgs( arg_ptr )
ARG_PTR arg_ptr ;
{
    if( arg_ptr != NULL ) {
        PutArg( arg_ptr ) ;
        while( arg_ptr->next_arg != NULL ) {
            arg_ptr = arg_ptr->next_arg ;
            PutArg( arg_ptr ) ;
        }
    }
    return ;
}

void PutArg( arg_ptr )
ARG_PTR arg_ptr ;
{
    arg_ptr->in_use = FREE_ARG ;
    return ;
}

```

Listing: SetType.c

```

#include <WindowMgr.h>
#include <ListMgr.h>
#include <OSUtil.h>
#include <EventMgr.h>

#include "MacCalc.h"
#include "CalcData.h"
#include "Parser.h"

void RemoveLeadingBlanks( unsigned char * ) ;
void PackLine( unsigned char * ) ;

void SetType( cell_ptr, had_data )
CELL_PTR cell_ptr ;
int had_data ;
{
    if( cell_ptr->formula[0] != 0 ) {

```

```

RemoveLeadingBlanks( cell_ptr->formula );
if( cell_ptr->formula[1] == '=' ) { /* formula */
    cell_ptr->type = FORMULA ;
    PackLine( cell_ptr->formula );
}else{
    if( IsDigit( cell_ptr->formula[1] ) ||
        ( cell_ptr->formula[1] == '.' ) ) {
        cell_ptr->type = CONSTANT ;
    }else{
        cell_ptr->type = STRING ;
    }
}
}else{
    if( had_data ) {
        cell_ptr->type = CLEARED ;
    }else{
        cell_ptr->type = UNDEFINED ;
    }
}
return ;
}

void RemoveLeadingBlanks( formula )
unsigned char *formula ;
{
    int length ;
    unsigned char *str ;

    length = formula[0] ;
    str = &formula[1] ;
    while( ( *str == ' ' ) || ( *str == '\t' ) ) {
        str++ ;
        length-- ;
    }
    BlockMove( str, &formula[1], length ) ;
    formula[0] = length ;
    return ;
}

void PackLine( formula )
unsigned char *formula ;
{
    int old_length ;
    int new_length ;
    int i ;
    unsigned char *str ;
    Str255 buffer ;
    int str_flag = FALSE ;

    old_length = formula[0] ;
    new_length = 0 ;
    str = buffer ;
    for( i=1 ; i<= old_length ; i++ ) {
        if( ( ( formula[i] != ' ' ) || str_flag )
            && ( formula[i] != '\t' ) ) {
            *str = formula[i] ;
            if( formula[i] == '\\' ) {
                str_flag = str_flag ? FALSE:TRUE ;
            }
            new_length++ ;
            str++ ;
        }
    }
    BlockMove( buffer, &formula[1], new_length ) ;
    formula[0] = new_length ;
    return ;
}

```

Listing: SheetHndl.c

```

#include <ListMgr.h>
#include <WindowMgr.h>

#include "MacCalc.h"
#include "SheetHndl.h"

```

```

char column_header[MAX_COLUMNS] = {
    'A','B','C','D','E','F','G','H' ;
} ;
char row_header[MAX_ROWS] = { '1','2','3','4','5','6','7','8' } ;

void SetupRowandColumnHeader( ListHandle ) ;
void InitData( SHEET_WIN_HDL ) ;

int OpenNewSpreadsheet( title )
char *title ;
{
    WindowPtr new_window_ptr ;
    SHEET_WIN_HDL new_sheet_hdl ;
    Rect rView, dataBounds ;
    Point cSize ;
    int i ;

    /* variable initialization for screen */
    new_window_ptr = GetNewWindow( 128, NULL, (WindowPtr)-1L ) ;
    new_sheet_hdl
    =(SHEET_WIN_HDL)NewHandle(sizeof(SHEET_WIN));

    /* Set the window title */
    SetWTitle( new_window_ptr, title ) ;

    SetRect( &dataBounds, 0, 0, MAX_COLUMNS+1, MAX_ROWS+1 ) ;
    SetRect( &rView, new_window_ptr->portRect.left,
        new_window_ptr->portRect.top,
        new_window_ptr->portRect.right - 15,
        new_window_ptr->portRect.bottom - 15 ) ;

    SetPt( &cSize, CELL_WIDTH, CELL_HEIGHT ) ;

    (**new_sheet_hdl).sheet_list_hdl = LNew( &rView, &data-
        Bounds, cSize, 0, new_window_ptr, TRUE, TRUE, TRUE, TRUE );

    SetWRefCon( new_window_ptr, (long)new_sheet_hdl ) ;

    InitData( new_sheet_hdl ) ;

    (**new_sheet_hdl).sheet_window_ptr = new_window_ptr ;

    (**(**new_sheet_hdl).sheet_list_hdl).1ClickLoop =
    (Ptr)ClickLoop ;

    SetupRowandColumnHeader(
    (**new_sheet_hdl).sheet_list_hdl);

    ShowWindow( new_window_ptr ) ;

    return noErr ;
}

void SetupRowandColumnHeader( list_hdl )
ListHandle list_hdl ;
{
    Cell curr_cell ;
    int i ;

    i = 0 ;

    for( i = 1 ; i <= MAX_ROWS ; i++ ) {

        SetPt( &curr_cell, i, 0 ) ;
        LSetCell( &column_header[i-1], 1, curr_cell,
list_hdl);
        SetPt( &curr_cell, 0, i ) ;
        LSetCell( &row_header[i-1], 1, curr_cell, list_hdl ) ;
    }
    return ;
}

void InitData( sheet_record_hdl )
SHEET_WIN_HDL sheet_record_hdl ;
{

```



```

int i,j ;
for( i=0; i<MAX_ROWS;i++){
    for( j=0; j<MAX_COLUMNS;j++ ) {
        (**sheet_record_hdl).sheet_data[i][j].type = 0 ;
        (**sheet_record_hdl).sheet_data[i][j].value = 0L ;
        (**sheet_record_hdl).sheet_data[i][j].formula[0] =
0 ;
    }
}
return ;
}

```

Listing: Resources.r

```

*
* MacTutorCalc.R

MacTutorCalc.rsrc
???CALx

Type CALx = STR
    ,0
Spreadsheet Demo \0D0 by Bryan
Waters for MacTutor

* Multifinder Menu for Quit Cmd
Type mstr = STR
    ,100
File

* Multifinder Quit name
Type mstr = STR
    ,101
Quit

Type SIZE = GNRL
    ,-1
.H
08000    ;; $0800 = bits 11 set
.L
80000    ;; ( recommended)
.L
40000    ;; ( minimum)
.I

Type vers = GNRL
    , 1
.H
01      ;; byte vers # in BCD
10      ;; byte vers part 2 & 3
50      ;; byte release stage
$50=release
00      ;; byte stage of non-
release
00 00   ;; integer country 0=US
.P
1.1 (US)
.P
SpreadSheet Demo, 0 by Bryan
Waters for MacTutor 1989

Type vers = GNRL
    , 2
.H
05      ;; byte vers # in BCD
30      ;; byte vers part 2 & 3
50      ;; byte release stage
$50=release
00      ;; byte stage of non-
release
00 00   ;; integer country 0=US
.P
V5.3
.P

```

MacTutor Volume 5 Number 3

* ——— menus ———

```

Type MENU
    ,1 (0)
\14

Type MENU
    ,256 (0)
File
Quit

Type MENU
    ,257 (0)
Edit
Undo
(-
Cut
Copy
Paste
Clear

Type WIND
    ,128 (0)
MacCalc
43 20 202 419
Invisible GoAway
0
0

Type DLOG
    ,128 (32)
New Dialog
208 70 330 424
Visible NoGoAway
1
0
12258

Type DITL
    ,12258 (0)
4

Button
96 184 116 244
Enter

Button
96 264 116 324
Cancel

staticText Disabled
8 8 24 88
Enter data:

editText
8 96 80 328

Type DITL
    ,200 (0)

```

3

Button
56 160 76 220
OK

IconItem Disabled
8 16 40 48
128

staticText Disabled
24 56 40 224
Mismatched Parenthesis.

Type DITL
 ,201 (0)

3

Button
56 160 76 220
OK

IconItem Disabled
8 16 40 48
128

staticText Disabled
24 56 40 224
Invalid Number.

Type DITL
 ,202 (0)

3

Button
56 160 76 220
OK

IconItem Disabled
8 16 40 48
128

staticText Disabled
24 56 40 224
Invalid address.

Type DITL
 ,204 (0)

3

Button
56 160 76 220
OK

IconItem Disabled
8 16 40 48
128

staticText Disabled
24 56 40 224
Invalid function.

Type DITL
 ,203 (0)

3

Button
56 160 76 220
OK

IconItem Disabled
8 16 40 48
128

staticText Disabled
24 56 40 224
Address too large.

Type ALERT
 ,200 (32)
118 94 214 338
200
4444

Type ALERT
 ,201 (32)
118 94 214 338
201
4444

Type ALERT
 ,202 (32)
118 94 214 338
202
4444

Type ALERT
 ,204 (32)
118 94 214 338
204
4444

Type ALERT
 ,203 (32)
118 94 214 338
203
4444

Type ICON = GNRL
 ,128 (32)
.H
0001 8000 0003 C000 0003 C000
0006 6000
0006 6000 000C 3000 000C 3000
0018 1800
0019 9800 0033 CC00 0033 CC00
0063 C600
0063 C600 00C3 C300 00C3 C300
0183 C180
0183 C180 0303 C0C0 0303 C0C0
0603 C060
0601 8060 0C01 8030 0C00 0030
1800 0018
1801 8018 3003 C00C 3003 C00C
6001 8006
6000 0006 C000 0003 FFFF FFFF
7FFF FFFE



C Workshop

All About Scrolling Windows



John A. Nairn
Salt Lake City, UT
Volume 5, Number 4

A Scrolling Manager Why and How to Scroll

The Macintosh user interface is based on windows. Windows provide a solution to the problem of keeping track of much information on a single computer screen. Each window can be structured to contain a single document. By document, I mean a logically connected set of text, graphics, or text and graphics that provides the working environment for a particular application. By the process of activating a window which brings that window to the foreground, the user can work on the document in that window. While working in one window, the contents of deactivated windows are within easy reach - a mouse click away.

Because the physical size of a window is limited to the physical size of the computer screen, a common occurrence will be that the document is larger than the window. In this situation, the window will only be able to display a portion of the document. The window based interface would fail if it were not possible to relocate the window and to display different portions of the document. The task of relocating windows or rather of repositioning documents behind windows is accomplished by scroll bars. Two scroll bars are required - a vertical scroll bar for scrolling the window contents up or down and a horizontal scroll bar for scrolling the window contents left or right.

While scroll bars are an important user interface feature, like many other Macintosh features, their implementation is less than trivial. My first attempt at using scroll bars was a time consuming process. Aided by an excellent book (*C Programming Techniques for the Macintosh*TM by Z. R. Mednieks and T. M. Schilke) and after much trial and error, I was able to work out the kinks of window scrolling. My second attempt was much easier and remarkable similar to the first attempt. After a few more attempts, I decided it was time to generalize the scrolling routines to elevate inclusion of scrolling in applications to the trivial matter that it ought to be.

My efforts aimed at generalizing window scrolling have been developed into a scrolling manager. Using the scrolling manager and relatively few lines of code any window can be converted into a scrolling window. The window can have both vertical and horizontal scroll bars. The contents of the window can be graphic (i.e. application defined) or can contain a TextEdit record. Using the scrolling interface described in the next section, the scrolling manager supports tool bars and rulers. Additional features include automatic scrolling while selecting, drag scrolling, joy-stick scrolling, and automatic tracking of the selection point. This article will describe the development and use of the scrolling manager and illustrate its capabilities with a demonstration application.

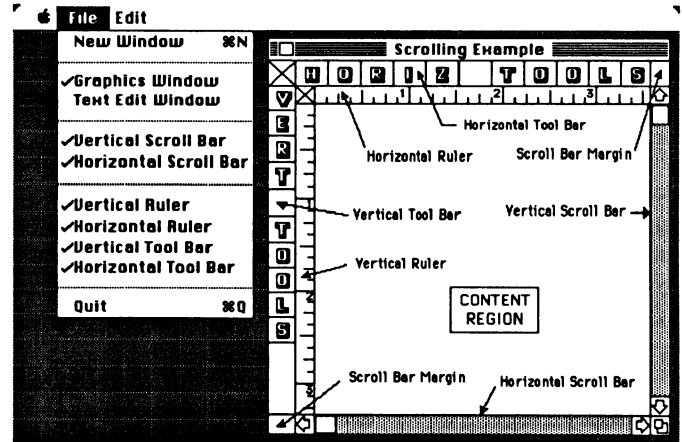


Figure 1. Scrollbars and Toolbars

A Scrolling Interface

The document window in the figure illustrates the important objects for the scrolling interface supported by the scrolling manager. The objects include vertical and horizontal scroll bars, tool bars, and rulers. The functions and the placement of each of these objects will be discussed in this section.

In virtually all applications, the horizontal scroll bar is along the bottom of the window with the right edge touching the size box in the lower right-hand corner of the window. Typically, the horizontal scroll bar extends the full width of the window with the left edge of the scroll bar touching the left edge of the window. Some applications stop the horizontal scroll bar before the left edge and leave a scroll bar margin which is used for some application-defined purpose (See figure). The scrolling interface defined by the scrolling manager supports horizontal scroll bars with one end touching the size box and the other end stopping an application-defined margin from the left edge of the window.

Vertical scroll bars are universally along the right edge of the window. In analogy with horizontal scroll bars, the scrolling manager supports vertical scroll bars whose bottom touches the size box and whose top stops at an application-defined margin from the top of the window.

The function of the scroll bars should be second nature to all experienced Macintosh users and will not be discussed at length here. In brief, the arrows scroll one line at a time, the gray regions between the arrows and the scroll box scroll one "page" (i.e. one window full) at a time and by dragging the scroll box, you can scroll to any position in the document.

Some applications (e.g. MacDrawTM) incorporate tool bars along the edge of their windows. The tool bar areas are static areas that do not scroll as the contents of the rest of the window scrolls. While the most common use of tool bars will be to

provide tools for the user, they can be used to display any static information that is meaningful to a particular application.

The scrolling manager supports vertical and/or horizontal tool bars. The vertical tool bar is a static area of application-defined width along the left edge of the window extending from the top of the window to the top of the horizontal scroll bar. The horizontal tool bar of application-defined height is along the top of the window and extends from the left edge of the window to the left edge of the vertical scroll bar.

Another common feature of in windows is rulers. Rulers are typically used in graphics applications to mark and measure position within a drawing. When scrolling occurs, the rulers should sometimes scroll and sometimes remain stationary. Consider for example a vertical ruler. As the window is scrolled in the vertical direction, the ruler should scroll along with the window contents. But, when horizontal scrolling occurs, the ruler should remain stationary otherwise it might scroll out of view. Likewise, horizontal rulers should scroll in the horizontal direction but remain stationary during vertical scrolling.

The scrolling manager supports vertical and/or horizontal rulers. The vertical ruler is of application-defined width and is just to the right of the vertical tool bar. The horizontal ruler is of application-defined height and is just below the horizontal tool bar.

The contents of rulers are not restricted to ruler markings. The ruler area, for example, might also include row and column labels in spreadsheet software. In terms of scrolling, row and column labels are logically equivalent to vertical and horizontal rulers. During vertical or row scrolling, row labels, just as vertical rulers, should scroll along with the window contents. During horizontal or column scrolling, the row labels, again just as vertical rulers, should remain stationary.

The last scrolling object is the content region which is the area of the window left for the document after removing the areas occupied by the size box, the scroll bars, the tool bars, and the rulers. In any scrolling window, the content region will be rectangular.

The scrolling objects defined in this section comprise a workable user-interface for scrolling windows. We note that the windows in virtually all Macintosh applications can be described by using some subset of these scrolling objects. The scrolling in all these applications could therefore be handled using the scrolling manager.

Scrolling in some applications could not be handled with the scrolling manager because those applications select "unusual" locations for scrolling objects. In Adobe™ Illustrator, for example, the rulers appear adjacent to the scroll bars rather than adjacent to the tool bars. Rulers in the position I described, however, would be equally functional to rulers located elsewhere. I therefore see no compelling reason for choosing a variety of ruler locations in a variety of different applications. It can be argued instead, that the user should be provided with a consistent scrolling interface. By providing specific locations for scroll bars, tool bars, and rulers, the scrolling manager attempts to provide a consistent scrolling interface.

The scrolling manager was developed to be stable with

respect to all subsets of scrolling objects. Although this stability includes subsets lacking one or even both scroll bars, I view the use of less than the full complement of scroll bars a violation (albeit a mild one) of the Macintosh user interface. When an application does not provide scroll bars, the user is limited in his options as to the size of the document window. If the window is made too small, the user will not be able to view the entire document without the inconvenience of resizing the window. In short the user's desktop ceases to be efficiently customizable to fit all user's needs. Before MultiFinder, limitations on window sizes were not very severe. But under MultiFinder, with the possibility of many open windows, the user may have valid reasons for resizing windows in ways not envisioned by developers of specific applications. I therefore suggest that all document windows, no matter what their contents, should include both vertical and horizontal scroll bars.

ScrollInfo Record

To operate window scrolling, the scrolling manager needs to know which scroll bars to include, the size and whether or not to include tool bars and rulers, and some other information related to scrolling. This information is contained in a data structure of type **ScrollInfo** which is defined as follows:

```
typedef struct {
    ControlHandle hScrollHdl;
    int horizLines;
    int horizLinesVis;
    Rect horizScrollRect;
    Point hRectTopLeft;
    int hScrollMarg;
    ControlHandle vScrollHdl;
    int vertLines;
    int vertLinesVis;
    Rect vertScrollRect;
    Point vRectTopLeft;
    int vScrollMarg;
    TEHandle hTE;
    long refCon;
} ScrollInfo, *ScrollPtr;
```

The **hScrollHdl** and **vScrollHdl** fields are handles for the scroll bars. Although as discussed in the introduction, it is recommended that all windows use both scroll bars, the scrolling manager will support windows with one or even with no scroll bars. When a scroll bar is absent, its handle field is set to NIL.

The **horizLines** and **vertLines** fields specify the total number of lines in the document. This information is needed to set the range of the scroll bars.

The **horizLinesVis** and **vertLinesVis** fields give the number of lines currently visible in the window. When the window is resized, these fields are adjusted accordingly.

The scrolling manager will also need to know the vertical and horizontal line size in number of pixels per line. This line size is used to calculate the number of lines in the document and the number of lines currently visible. The line size is also used to decide the number of pixels that should be scrolled when the user clicks in the scroll bar arrows and therefore calls for scrolling one line at a time.

If the line size is too small, scrolling will be smooth but might

be too slow. If the line size is too large, scrolling will be fast but might be jerky. A happy medium can usually be found with line sizes between 8 and 16 pixels. For vertical scrolling of text windows, it make sense to set the vertical line size to the number of pixels in one line of text.

The scrolling manager stores the vertical and horizontal line sizes in the `refCon`'s of the scroll bars. The horizontal line size is stored as minus the horizontal line size. With this scheme, by checking the sign of a scroll bar's `refCon`, the scrolling manager can determine if a given control handle corresponds to a horizontal or to a vertical scroll bar.

When the user initiates scrolling, the scrolling manager eventually calls the QuickDraw routine `ScrollRect()` which scrolls a specified rectangle by a specified number of pixels in the vertical or horizontal directions. The `horizScrollRect` and `vertScrollRect` fields are the rectangles passed to `ScrollRect()` during horizontal or vertical scrolling, respectively. The `horizScrollRect` will include the content region and the horizontal ruler. The `vertScrollRect` will include the content region and the vertical ruler. For windows which contain rulers, these two rectangles will be different.

The `hRectTopLeft` and `vRectTopLeft` fields are used to define the scrolling rectangles and they give the top-left corner of the rectangles. The bottom-right corner of the scrolling rectangles will be at the top-left corner of the size box when both scroll bars are being used and will extend to the window edge when one of the scroll bars is missing. These fields also determine the presence or absence and the size of tool bars and rulers.

The `vScrollMarg` and `hScrollMarg` fields contain the scroll bar margins illustrated in the figure.

For text windows that use the Text Edit manager, the field `hTE` holds the handle to the Text Edit record.

Finally, the `refCon` field is provided for application-defined contents. A logical use of the `refCon` and the reason for its inclusion are described latter.

Using The Scrolling Manager

Typical event-driven applications have sections of code that handle common events such as activating, deactivating, and updating windows or responding to key down or mouse events. Within these or other common sections of code, simple calls to the scrolling manager will take care of all scrolling.

When a new scrolling window is created, call `SetScrollWindow()` which will return a pointer to the `ScrollInfo` record. This pointer will need to be passed to many of the scrolling manager routines.

The procedures `ScrollActivate()` and `ScrollDeactivate()` must be called whenever an activate or a deactivate event is reported for a scrolling window. These routines take care of highlighting or unhighlighting the scroll bars and also set an internal `ScrollInfo` record pointer which points to the current active window's `ScrollInfo` record. The internal pointer is used by routines which apply only to the currently active window.

Whenever an active window is resized call `GrowScroll()` which moves and resizes the scroll bars. Before and immediately after resizing the window using `SizeWindow()`, call `InvalBars()`

which invalidates the scroll bar rectangles to insure that the old scroll bars are erased and that the new scroll bars are correctly drawn.

The section of code which updates window contents must be written to be accessible by a call using

```
UpdateWindow(upWind,scrollType)
```

where `upWind` is a pointer to the window requiring updating and `scrollType` will be equal to `VertBar` (1) or `HorizBar` (2) when called from the scrolling manager and should be `FALSE` (0) when called from your application. The way applications should use `scrollType` is discussed latter.

Whenever the document changes size in either the vertical or horizontal directions, call `SetScroll()`. This routine will reset the appropriate variable `vertLines` or `horizLines` and will reset the appropriate scroll bar's maximum value.

When a mouse down event in the content of a window is reported call `DoScroll()`. If `DoScroll()` returns `FALSE` then handle the mouse down event as appropriate. If `DoScroll()` returns `TRUE` then the mouse down event was a scrolling event that was handled by the scrolling manager and no further action needs to be taken.

When the user is making a selection of text or of graphics and drags the mouse outside the window, it is desirable to implement autoscrolling to enable the user to make a selection that is larger than the window. Implementing autoscrolling in Text Edit windows is easy - merely replace the standard call to `TEClick()` with a call to `ScrollTEClick()`. `ScrollTEClick()` calls `TEClick()` from a section of code that will always be in the same code segment as the click loop routine provided by the scrolling manager.

Implementing autoscrolling in graphic windows is not much harder. Whenever the application believes that autoscrolling may be necessary, a call to `FollowMouse()` will scroll the window in the direction of the mouse if the mouse has been dragged out of the content region. An example of graphic autoscrolling while drawing a polygon is given in the routine `TrackPoly()` in the listing of `Scroller`.

A second need for autoscrolling arises when the current selection is not visible in the window and the user performs some operation on it. *Inside Macintosh* recommends that applications scroll the selection into view before performing the operation. The two routines `FixInsertPt()` and `FixTEInsertPt()` scroll the insertion point of the active window into view. The latter brings the insertion point in Text Edit windows into view. If no place else, `FixTEInsertPt()` should be called after every key down event in Text Edit windows. This call will insure that the window scrolls as the user types beyond the bottom of the window.

It might be desirable to provide the user with alternative means to scrolling. Two such means in the scrolling manager are drag scrolling and joy-stick scrolling. Drag scrolling is available in many applications as a hand tool. As the mouse is clicked and dragged using the hand tool, the window is dragged along with it. Using the scrolling manager, when a mouse down event is reported and drag scrolling should follow, call `DragScroll()`.

In joy-stick scrolling, the mouse is clicked at some point and dragged in any direction. The window is scrolled in the direction that the mouse is dragged and is scrolled faster the farther the mouse is dragged from the original point. Using the scrolling manager, when a mouse down event is reported and joy-stick scrolling should follow, call **JoyStickScroll()**. The joy-stick sensitivity, or how far the mouse must be dragged from the initial point to increase scrolling speed, is set by calling **SetJoySpeed()**.

When a scrolling window is closed and the **ScrollInfo** record was allocated in the heap, call **DisposeScroll()**.

A number of routines in the scrolling manager are provided as useful utilities:

GetMargins(): returns distance of the top-left corner of the window to the top-left corner of the document in pixels or in scrolling lines. This routine should probably be called from your update routine to determine which part of the document should be drawn.

ScrollSectRect(): returns a pointer to a rectangle which is the intersection of the **vertScrollRect** and the **horizScrollRect** and will usually define the content region.

ScrollSectRgn(): same as **ScrollSectRect()** except returns a handle to a region. If the window has no scroll bars, the region will also exclude the size box.

NonScrollRect(): returns pointer to rectangle that excludes the window's scroll bars.

ScrollDrawGrowIcon(): replaces **Toolbox DrawGrowIcon()** and only draws the parts of the grow icon needed to outline the window's scroll bars.

GetSRefCon(): returns the **refCon** in the **ScrollInfo** record.

SetSRefCon(): sets the **refCon** in the **ScrollInfo** record.

ActiveScrollPtr(): returns the **ScrollPtr** to the current active scrolling window.

Any routines in the scrolling manager not discussed in this section are intended for internal use.

As a bottom line, the scrolling manager does not require much work to implement. The most optimal situation would be a document that never changes size. Scrolling in this window would only take 7 lines of code. These lines would be calls to **SetScrollWindow()**, **ScrollActivate()**, **ScrollDeactivate()**, **GrowScroll()**, **DoScroll()**, and two calls to **InvalidBars()**. Adding additional scrolling features such as variable size documents, autoscrolling, drag scrolling, and joy-stick scrolling will only require a few more lines of code.

Sample Application: Scroller

The source code for the application Scroller follows this article. The first half of the listing is the header and the source code for the scrolling manager. The second half is the source code for Scroller which functions to demonstrate the features of the scrolling manager.

Scroller has only two menus - File and Edit menus. Under the File menu are the various settings for the features in the scrolling window (see figure). The window can be a Text Edit window or a graphics window. The window can have tool bars and/or rulers. The window can have both scroll bars, only one scroll bar, or no scroll bars. Selecting one of the File menu items will check that

menu item and incorporate that scrolling feature into the next scrolling window.

Once the desired scrolling features are checked, choose "New Window" and a scrolling window incorporating those features will be created. Only one window will ever be present in Scroller. If "New Window" is selected and a window is currently open, that current window will first be closed. Note also, that selecting scrolling features in the File menu will only take effect on the next window and will not effect the current window if one is open.

Graphic windows will open with a blank drawing that is ten inches square. For demonstration purposes, the mouse will function as a polygon tool. On each click and drag, a line segment will be drawn from the previous polygon point to the current mouse location. If you click and drag outside the content region, the window will autoscroll until it reaches the edge of the drawing.

Text windows will open with a paragraph of text. You can type into this window and cut and paste text using standard Macintosh text handling procedures. The Edit menu, which has the standard editing commands, will be enabled for Text Edit windows.

One feature not accessible in the File menu are the scroll-bar margins. To illustrate use of scroll-bar margins, Scroller will set the margins to the width of the tool bars whenever tool bars are present. In actual use, it is of course possible to set the scroll-bar margins to any value independent of the width of the tool bars.

Finally, drag scrolling and joy-stick scrolling can also be tried using Scroller. To scroll using drag scrolling hold down the option key. To scroll using joy-stick scrolling hold down the shift and the option keys. For drag scrolling the cursor changes to the standard hand shape. For joy-stick scrolling, the cursor changes to four small arrows pointing up, down, left, and right.

Technical Comments

The most important sections of the scrolling manager are the routines that actually do the scrolling. All scrolling, whether scroll-bar scrolling, drag scrolling, joy-stick scrolling, or autoscrolling is eventually handled by the internal routine **ScrollWindow(theWindow, units)** which scrolls the window pointed to by **theWindow** by **units** lines. Before calling **ScrollWindow()** the variables **scrip** (a pointer to the active window's **ScrollInfo** record) and **scrollType** (equal to **VertBar** (1) or **HorizBar** (2)) will have already been set. The way these variable get set will depend on the type of scrolling occurring.

In brief, **ScrollWindow()** will call **ScrollRect()** for graphics windows and call **TEScroll()** for Text Edit windows. If a Text Edit window has rulers than a call to **ScrollRect()** will be required to scroll the contents of the rulers. For purely Text Edit windows, the call to **TEScroll()** completes the job. Apparently **TEScroll()** calls **TEUpdate()** and therefore automatically updates the text. Graphics windows or Text Edit windows with rulers will have some application-defined contents. To update the application-defined portions of these windows that may have been disturbed by scrolling, **ScrollWindow()** ends by calling **UpdateWindow()** which as discussed above must be supplied by

the application. Note that the application supplied **UpdateWindow()** routine must check the current scroll bar margins to draw the correct part of the document. The routine **GetMargins()** is provided to simplify this task.

The scrolling speed is determined by a combination of number of pixels per line and the efficiency of the scrolling manager routines. For a given line size, it turns out that the limiting factor in scrolling speed is the speed at which the application supplied routine **UpdateWindow()** can update the window contents. The variable **scrollType** is supplied to **UpdateWindow()** to help in updating efficiency. When **UpdateWindow()** is called from **ScrollWindow()**, **scrollType** will be equal to **VertBar** (1) or **HorizBar** (2) at which time only the contents of the **vertScrollRect** or of the **horizScrollRect** respectively will need to be redrawn. When **UpdateWindow()** is called from the application **scrollType** should be **FALSE** (0) and the whole window may need to be redrawn.

Because the efficiency of **UpdateWindow()** is crucial, applications that use rulers should use clever means of quickly drawing them. The application **Scroller** does not use a clever ruler scheme and the slow down in scrolling between windows with and without rulers is apparent. The techniques used for updating the tool bars are not important because they need never be called while scrolling.

With *Inside Macintosh Volume IV* autoscrolling was implemented into the Text Edit routines. Because the Text Edit autoscrolling routines are not aware of the details of the window's scrolling interface, they were not of use in developing the scrolling manager. For Text Edit windows which have rulers, the scrolling manager will have to handle autoscrolling of the rulers anyway, so it might as well do the text autoscrolling at the same time.

To handle autoscrolling while selecting, the scrolling manager provides a default click loop called **SMClickLoop()**. The click loop routine merely checks the mouse location and scrolls one line in the appropriate direction if it is outside the Text Edit view rectangle. One complicating aspect of developing the click loop is that **TEClick()** which calls the click loop apparently sets the clip region to the view rectangle. The click loop routine must reset the clip region to the entire window otherwise the scroll bar graphics and the rulers will not scroll appropriately during autoscrolling. The click loop routine also better restore the clip region before returning to **TEClick()** or else the inverting of the selected text will not occur correctly. I would have been saved several hours of work had *Inside Macintosh* included a warning regarding the clip region and **TEClick()**.

The hardest part of Text Edit autoscrolling was the routine **FixTEInsertPt()** which scrolls the beginning of the selection region into view. This routine is typically called whenever **TEKey()**, **TECut()**, **TEDelete()**, or **TEPaste()** have been called. Calls to **FixTEInsertPt()** will become frequent while the user is typing in text and it therefore must be fast.

FixTEInsertPt() begins with a binary search through the **lineStarts** array in the Text Edit record to find the line which has the start of the current selection. The search begins at the last line. By starting on the last line **FixTEInsertPt()** will be fastest during

the most common form of typing which is adding new text at the end of document. Even for large Text Edit documents (which are limited to 32K) a binary search to any location in the document is fast. Frequent calls to **FixTEInsertPt()** will therefore not result in any response problems while entering text.

I like to think part of the speed of **FixTEInsertPt()** was a result of careful attention to the code for the binary search. Experienced C programmers could certainly see ways of shortening the code and making it look more elegant. My goal, however, was not to create concise C code but to optimize the machine code generated by **LightSpeed C™**. To achieve this goal, I used **MacBugs** to examine the machine code generated for the binary search. The C code was then modified until I ended up with the result given in the attached source code. The final machine code had about half the number of lines per loop and therefore is probably about twice as fast as the machine code resulting from more concise C code. Some of the key changes were to set up pointers before entering the search loop, use register variables wherever possible, and use logical operations (e.g. shifts) instead of arithmetic expressions wherever possible.

Note that **FixTEInsertPt()** assumes constant line heights as was always the case for original Text Edit records. With the recent implementation of various text attributes within a single Text Edit record, **FixTEInsertPt()** will have to be modified. The changes will be relatively minor and I plan to get to it as soon as I need that feature (or maybe even sooner).

In multiwindow applications, the application usually will not be concerned with what window is scrolling. Many of the scrolling interface sections of code can therefore be generalized to handle any window. One thing the application will probably need to know will be the pointer to the **ScrollInfo** record for each window. This pointer can be stored in the windows **refCon** and therefore always be available. For this strategy not to cause the loss of other uses for the window's **refCon**, a replacement **refCon** is provided in the **ScrollInfo** record.

Scrolling Manager Routines

The useful routines in the scrolling manager are described in the section "Using the Scrolling Manager." The calling procedures; that is, the type and description of the required parameters for each routine, are contained in the comments in the source code.

Listing: ScrollMgr.h

```
/*Header file for scrolling manager
 * 1988 John A. Nairn, All Rights Reserved */

#ifndef _ControlMgr_
#include <ControlMgr.h>
#endif
#ifndef _TextEdit_
#include <TextEdit.h>
#endif
#ifndef _MacTypes_
#include <MacTypes.h>
#endif

#define SBarWidth 15/*Standard Constants*/
#define HorizBar 1
```

```
#define VertBar 2
#define inPixels 0
#define inLines 1

typedef struct (
    ControlHandle hScrollHdl;
    int horizLines;
    int horizLinesVis;
    Rect horizScrollRect;
    Point hRectTopLeft;
    int hScrollMarg;
    ControlHandle vScrollHdl;
    int vertLines;
    int vertLinesVis;
    Rect vertScrollRect;
    Point vRectTopLeft;
    int vScrollMarg;
    TEHandle hTE;
    long refCon;
) ScrollInfo, *ScrollPtr;
```

```
ScrollPtr SetScrollWindow();
RgnHandle ScrollSectRgn();
Rect *NonScrollRect();
Rect *ScrollSectRect();
long GetSRefCon();
ScrollPtr ActiveScrollPtr();
```

Listing: Scroll.c

```
/* Scrolling Manager Library of Subroutines
 * 1988 John A. Nairn, All Rights Reserved
```

This library of routines allow any application to easily implement scrolling in any window. The library can handle application-defined graphic windows as well as Text Edit windows.

General variables appearing as common parameters

1. theScrlp: the ScrollPtr for scrolling window
2. theWindow: window pointer to scrolling window

Routines marked "internal use" should not need to be called by the application

WARNING: A number of these routines assume scrolling is occurring in the active window. This means that ScrollActivate() must have been called before calling these routines. Failure to do so may result in system crash.

```
*/
```

```
#include <MacTypes.h> /*Includes */
#include <QuickDraw.h>
#include <ControlMgr.h>
#include <TextEdit.h>
#include "ScrollMgr.h"
#define Active 0 /* defines */
#define Inactive 255
#define NIL 0L
```

```
ScrollPtr scrlp; /* Global Variables */
int scrollAmt,scrollCode,scrollType,linesVis,lineSize;
int vJoySpeed=8;hJoySpeed=8;
pascal Boolean SMClikLoop();
```

```
/* Initialize ScrollInfo data record for new window and
return ScrollPtr as the function return. Note
this routine calls ScrollActivate() to activate
the scroll bars.
```

1. theScrlp: pointer to use for ScrollInfo or NIL to put ScrollInfo in heap
2. textWindow=TRUE if Text Edit window else =FALSE
3. v or hLineSize: pixels per line to scroll (0 to

- not include scroll bar)
4. v or hTotalLines: number of lines in document
5. v or hTools: width of tool bars in pixels (0 if no tool bar)
6. v or hRuler: width of rulers in pixels (0 if no ruler)
7. vScrollTop or hScrollLeft: scroll bar margins
8. sRefCon: ScrollInfo refCon */

```
ScrollPtr SetScrollWindow(theWindow,theScrlp,textWindow,
    vLineSize,vTotalLines,vTools,vRuler,
    vScrollTop,hLineSize,hTotalLines,hTools,
    hRuler,hScrollLeft,sRefCon)
```

```
WindowPtr theWindow;
ScrollPtr theScrlp;
Boolean textWindow;
int vLineSize,vTotalLines,vTools,vRuler;
int vScrollTop,hLineSize,hTotalLines,hTools;
int hRuler,hScrollLeft;
long sRefCon;
```

```
(
    ScrollPtr newscrlp;
    Rect viewRect,scrollRect;

    if(theScrlp==NIL)
        newscrlp=(ScrollInfo *)
            NewPtr((Size)sizeof(ScrollInfo));
    else
        newscrlp=theScrlp;
```

```
/*vertical scroll bar initialization*/
newscrlp->vScrollMarg=vScrollTop;
newscrlp->vRectTopLeft.v=hTools+hRuler;
newscrlp->vRectTopLeft.h=vTools;
if(vLineSize!=0)
    ( SetVertRect(newscrlp,theWindow,vLineSize,
        hLineSize);
    scrollRect=theWindow->portRect;
    scrollRect.left=scrollRect.right-SBarWidth;
    scrollRect.right+=1;
    scrollRect.bottom-=14;
    scrollRect.top-=1-newscrlp->vScrollMarg;
    newscrlp->vScrollHdl=NewControl(theWindow,
        &scrollRect,"v",1,0,0,0,scrollBarProc,
        (long)vLineSize);
    SetScroll(newscrlp,vTotalLines,VertBar);
)
else
    newscrlp->vScrollHdl=NIL;
```

```
/*horizontal scroll bar initialization*/
newscrlp->hScrollMarg=hScrollLeft;
newscrlp->hRectTopLeft.v=hTools;
newscrlp->hRectTopLeft.h=vTools+vRuler;
if(hLineSize!=0)
    ( SetHorizRect(newscrlp,theWindow,hLineSize,
        vLineSize);
    scrollRect=theWindow->portRect;
    scrollRect.left-=1-newscrlp->hScrollMarg;
    scrollRect.right-=SBarWidth-1;
    scrollRect.top=scrollRect.bottom-SBarWidth;
    scrollRect.bottom+=1;
    newscrlp->hScrollHdl=NewControl(theWindow,
        &scrollRect,"h",1,0,0,0,scrollBarProc,
        (long)(-hLineSize));
    SetScroll(newscrlp,hTotalLines,HorizBar);
)
else
    newscrlp->hScrollHdl=NIL;
```

```
/*Text edit initialization (if text window) */
```

```
if(textWindow)
    ( ScrollSectRect(newscrlp,theWindow,&viewRect);
```

```

    newscrlp->hTE=TENew(&viewRect,&viewRect);
    SetClickLoop(&SMClickLoop,newscrlp->hTE);
}
else
    newscrlp->hTE=NIL;

newscrlp->refCon=sRefCon;
ScrollActivate(newscrlp);
return(newscrlp);
}

/*Set the scrolling rectangles - internal use */
SetVertRect(newscrlp,theWindow,theLineSize,hasHorizBar)
ScrollPtr newscrlp;
WindowPtr theWindow;
int theLineSize,hasHorizBar;
{
    Rect r;

    r=theWindow->portRect;
    r.left+=newscrlp->vRectTopLeft.h;
    r.top+=newscrlp->vRectTopLeft.v;
    r.right-=SBarWidth;
    if(hasHorizBar)
        r.bottom-=SBarWidth;
    newscrlp->vertScrollRect=r;
    newscrlp->vertLinesVis=(r.bottom-r.top)/theLineSize;
}

SetHorizRect(newscrlp,theWindow,theLineSize,hasVertBar)
ScrollPtr newscrlp;
WindowPtr theWindow;
int theLineSize,hasVertBar;
{
    Rect r;

    r=theWindow->portRect;
    r.left+=newscrlp->hRectTopLeft.h;
    r.top+=newscrlp->hRectTopLeft.v;
    r.bottom-=SBarWidth;
    if(hasVertBar)
        r.right-=SBarWidth;
    newscrlp->horizScrollRect=r;
    newscrlp->horizLinesVis=(r.right-r.left)/theLineSize;
}

/*Call when window activates: will activate scroll bars
and set internal pointer scrip to newscrlp which
is ScrollPtr for window being activated */
ScrollActivate(newscrlp)
ScrollPtr newscrlp;
{
    scrip=newscrlp;
    if(scrip->vScrollHdl!=NIL)
        HiliteControl(scrip->vScrollHdl,Active);
    if(scrip->hScrollHdl!=NIL)
        HiliteControl(scrip->hScrollHdl,Active);
}

/*Call when window deactivates: will deactivate scroll
bars. oldscrip is ScrollPtr for deactivating
window */
ScrollDeactivate(oldscrip)
ScrollPtr oldscrip;
{
    if(oldscrip->vScrollHdl!=NIL)
        HiliteControl(oldscrip->vScrollHdl,Inactive);
    if(oldscrip->hScrollHdl!=NIL)
        HiliteControl(oldscrip->hScrollHdl,Inactive);
}

```

```

/*Call when scrolling window changes size.
Note: assumes growing window is active window */
GrowScroll(theWindow)
WindowPtr theWindow;
{
    int theLineSize,hasOtherBar;
    int newTop,newRight,newBottom,newLeft;
    Rect tempRect;

    if(scrip->vScrollHdl!=NIL)
    { /*Move vertical Bar */
        newTop=theWindow->portRect.top-1+
            scrip->vScrollMarg;
        newRight=theWindow->portRect.right-SBarWidth;
        newBottom=theWindow->portRect.bottom-SBarWidth;
        MoveControl(scrip->vScrollHdl,newRight,newTop);
        SizeControl(scrip->vScrollHdl,SBarWidth+1,
            newBottom-newTop+1);

        /*Reset vertical scrolling rectangle */
        theLineSize=GetCRefCon(scrip->vScrollHdl);
        hasOtherBar=(scrip->hScrollHdl!=NIL)?TRUE:FALSE;
        SetVertRect(scrip,theWindow,theLineSize,
            hasOtherBar);

        /*Reset vertical scroll bar range */
        SetScroll(scrip,scrip->vertLines,VertBar);
    }

    if(scrip->hScrollHdl!=NIL)
    { /*Move horizontal Bar */
        newTop=theWindow->portRect.bottom-SBarWidth;
        newLeft=theWindow->portRect.left-1+
            scrip->hScrollMarg;
        newRight=theWindow->portRect.right-SBarWidth;
        MoveControl(scrip->hScrollHdl,newLeft,newTop);
        SizeControl(scrip->hScrollHdl,newRight-newLeft+1,
            SBarWidth+1);

        /*Reset vertical scrolling rectangle */
        theLineSize=-GetCRefCon(scrip->hScrollHdl);
        hasOtherBar=(scrip->vScrollHdl!=NIL)?TRUE:FALSE;
        SetHorizRect(scrip,theWindow,theLineSize,
            hasOtherBar);

        /*Reset vertical scroll bar range */
        SetScroll(scrip,scrip->horizLines,HorizBar);
    }

    /*Reset view rectangle (if text window) */
    if(scrip->hTE!=NIL)
    { ScrollSectRect(scrip,theWindow,&tempRect);
      (**scrip->hTE).viewRect=tempRect;
    }

    /*Invalidate the scroll bar area before and after
    window resizing */

    InvalBars(theWindow)
    WindowPtr theWindow;
    {
        Rect r;

        r=theWindow->portRect; /*Horizontal ScrollBar*/
        r.top=r.bottom-SBarWidth;
        InvalRect(&r);
        r.top=theWindow->portRect.top; /*Vertical Scroll Bar*/
        r.left=r.right-SBarWidth;
        InvalRect(&r);
    }
}

```



```

/*Reset a scroll bar range when the document changes
size. Updates window if necessary.
1. totalLines: total Lines in document
2. barType: VertBar or HorizBar*/

SetScroll(theScrlp, totalLines, barType)
ScrollPtr theScrlp;
int totalLines, barType;
(
int n, max, currentValue, numLinesVis, tempLineSize, units;
WindowPtr ctrlWindow;
ControlHandle scrollHdl;

if(barType==VertBar)
( tempLineSize=GetCRefCon(theScrlp->vScrollHdl);
numLinesVis=theScrlp->vertLinesVis;
scrollHdl=theScrlp->vScrollHdl;
theScrlp->vertLines=totalLines;
)
else
( tempLineSize=-GetCRefCon(theScrlp->hScrollHdl);
numLinesVis=theScrlp->horizLinesVis;
scrollHdl=theScrlp->hScrollHdl;
theScrlp->horizLines=totalLines;
)

n=totalLines-numLinesVis;
currentValue=GetCtlValue(scrollHdl);
max=(n>0 ? n : 0);
SetCtlMax(scrollHdl, max);

/* if necessary, scroll and redraw the window */
if(currentValue>max)
( ctrlWindow=(*scrollHdl)->ctrlOwner;
InvalRect(&ctrlWindow->portRect);
if(theScrlp->hTE!=NIL)
( units=tempLineSize*(currentValue-max);
if(barType==VertBar)
OffsetRect(&(*theScrlp->hTE).destRect,
0, units);
else
OffsetRect(&(*theScrlp->hTE).destRect,
units, 0);
)
)

/*Return current margins in pixels or in scrolling
lines for window.
1. left or topMargin: margin returned in
these variables
2. lm or tmType: set to inPixels (0) or
inLines (1) for result in pixels
or lines */

GetMargins(theScrlp, leftMargin, lmType, topMargin, tmType)
ScrollPtr theScrlp;
int *leftMargin, *topMargin, lmType, tmType;
(
int theLineSize;

if(theScrlp->vScrollHdl!=NIL)
( *topMargin=GetCtlValue(theScrlp->vScrollHdl);
if(tmType==inPixels)
( theLineSize=GetCRefCon(theScrlp->vScrollHdl);
*topMargin*=theLineSize;
)
)
else
*topMargin=0;
if(theScrlp->hScrollHdl!=NIL)
( *leftMargin=GetCtlValue(theScrlp->hScrollHdl);
if(lmType==inPixels)
( theLineSize=-GetCRefCon(theScrlp->hScrollHdl);

```

```

*leftMargin*=theLineSize;
)
)
else
*leftMargin=0;
)

/*Find intersection of two scrolling rectangles, if no
scroll bars, this rect will include the grow box
1. drawRect: pointer to resulting rectangle - this
pointer also returned as function value */

Rect *ScrollSectRect(theScrlp, theWindow, drawRect)
ScrollPtr theScrlp;
WindowPtr theWindow;
Rect *drawRect;
(
NonScrollRect(theScrlp, theWindow, drawRect);
drawRect->top=theScrlp->vRectTopLeft.v;
if(theScrlp->hRectTopLeft.v>drawRect->top)
drawRect->top=theScrlp->hRectTopLeft.v;
drawRect->left=theScrlp->vRectTopLeft.h;
if(theScrlp->hRectTopLeft.h>drawRect->left)
drawRect->left=theScrlp->hRectTopLeft.h;
return(drawRect);
)

/*Return handle for region that is intersection of two
scrolling rectangles. If no scroll bars, region will
exclude grow box */

RgnHandle ScrollSectRgn(theScrlp, theWindow)
ScrollPtr theScrlp;
WindowPtr theWindow;
(
Rect r;
RgnHandle drawRgn, tempRgn;

ScrollSectRect(theScrlp, theWindow, &r);
drawRgn=NewRgn();
RectRgn(drawRgn, &r);
if((theScrlp->vScrollHdl==NIL)&&
(theScrlp->hScrollHdl==NIL))
( tempRgn=NewRgn();
r.left=theWindow->portRect.right-SBarWidth;
r.top=theWindow->portRect.bottom-SBarWidth;
r.right=theWindow->portRect.right;
r.bottom=theWindow->portRect.bottom;
RectRgn(tempRgn, &r);
DiffRgn(drawRgn, tempRgn, drawRgn);
DisposeRgn(tempRgn);
)
return(drawRgn);
)

/*Calculate rectangle that excludes the scroll bars
1. drawRect: pointer to resulting rectangle - this
pointer also returned as function value */

Rect *NonScrollRect(theScrlp, theWindow, drawRect)
ScrollPtr theScrlp;
WindowPtr theWindow;
Rect *drawRect;
(
*drawRect=theWindow->portRect;
if(theScrlp->vScrollHdl!=NIL)
drawRect->right-=SBarWidth;
if(theScrlp->hScrollHdl!=NIL)
drawRect->bottom-=SBarWidth;
return(drawRect);
)

/*Draw enough grow icon to enclose active scroll bars */

```

```
ScrollDrawGrowIcon(theScrlp,theWindow)
```

```
    ScrollPtr theScrlp;
    WindowPtr theWindow;
```

```
    (
        Rect r;
        RgnHandle tempRgn;

        GetClip(tempRgn=NewRgn());
        r=theWindow->portRect;
        if(theScrlp->vScrollHdl==NIL)
            r.top=r.bottom-SBarWidth;
        if(theScrlp->hScrollHdl==NIL)
            r.left=r.right-SBarWidth;
        ClipRect(&r);
        DrawGrowIcon(theWindow);
        SetClip(tempRgn);
        DisposeRgn(tempRgn);
    )
```

```
/*Action procedure for scrolling - internal use
 Scroll by scrollAmt (set by doScroll)
 ScrollCode is the part of the scroll bar being tracked
 (set by doScroll) ScrollType is type of scroll bar
 (set by doScroll)
 May also be called for tracking selections. In this
 case scrollType will be set by setSelectScroll,
 scrollAmt will be 1 or -1 (set by selectScroll)
 and scrollCode will be set to 0. */
```

```
pascal void ScrollProc(control,theCode)
    ControlHandle control;
    int theCode;
```

```
    (
        int max,oldValue,newValue;
        WindowPtr ctrlWindow=(*control)->ctrlWindow;

        if(theCode==scrollCode)
        { oldValue=GetCtlValue(control);
          newValue=oldValue+scrollAmt;
          if(newValue<0) newValue=0;
          max=GetCtlMax(control);
          if(newValue>max) newValue=max;
          SetCtlValue(control,newValue);
          ThumbMove(ctrlWindow,oldValue,newValue);
        }
    )
```

```
/*Do scrolling, return TRUE if scrolled or FALSE if not
 1. where: mouse location in Global coordinates
 Note: assumes scrolling in current active window */
```

```
DoScroll(theWindow,where)
```

```
    WindowPtr theWindow;
    long where;
```

```
    (
        int partCode,oldValue,newValue;
        ControlHandle control;

        GlobalToLocal(&where);
        partCode=FindControl(where,theWindow,&control);
        if(partCode==0) /*mouse down not in scroll bar*/
            return(FALSE);
```

```
        lineSize=GetCRefCon(control);
        if(lineSize<0) /* horiz scrolling */
        { linesVis=scrlp->horizLinesVis;
          scrollType=HorizBar;
          lineSize=-lineSize;
        }
        else /* vert scrolling */
        { linesVis=scrlp->vertLinesVis;
          scrollType=VertBar;
        }
    )
```

```
switch(partCode)
```

```
    ( case inUpButton:
      case inDownButton:
      case inPageUp:
      case inPageDown:
        switch(partCode)
        { case inUpButton:
          scrollAmt=-1;
          break;
          case inDownButton:
          scrollAmt=1;
          break;
          case inPageUp:
          scrollAmt=-linesVis;
          break;
          case inPageDown:
          scrollAmt=linesVis;
          break;
        }
        scrollCode=partCode;
        TrackControl(control,where,&ScrollProc);
        break;
        case inThumb:
        oldValue=GetCtlValue(control);
        if(TrackControl(control,where,-1L))
        { newValue=GetCtlValue(control);
          ThumbMove(theWindow,oldValue,newValue);
        }
        break;
        default:
        break;
    )
    return(TRUE);
}
```

```
/*Set to prepare for automatic scrolling of scroll bars
 - internal use */
```

```
SetSelectScroll(control)
```

```
    ControlHandle control;
```

```
    (
        lineSize=GetCRefCon(control);
        if(lineSize<0) /* horiz scrolling */
        { linesVis=scrlp->horizLinesVis;
          scrollType=HorizBar;
          lineSize=-lineSize;
        }
        else /* vert scrolling */
        { linesVis=scrlp->vertLinesVis;
          scrollType=VertBar;
        }
        scrollCode=0;
    )
```

```
/*Scroll dir lines using scroll bar set by
 setSelectScroll - internal use */
```

```
SelectScroll(dir)
```

```
    int dir;
```

```
    (
        scrollAmt=dir;
        if(scrollType==HorizBar)
            ScrollProc(scrlp->hScrollHdl,0);
        else
            ScrollProc(scrlp->vScrollHdl,0);
    )
```

```
/*Scroll along with mouse while mouse button is down
 1. where: mouse location in Global coordinates
 Note: assumes scrolling in current active window */
```

```
DragScroll(where)
```

```
    Point where;
```

```
    (
```

```

WindowPtr theWindow;
register int vLineSize=0,hLineSize=0,vAmt,hAmt;
Rect dragRect;
Point lastWhere;

if(scr1p->vScrollHdl!=NIL)
{ theWindow=(*(scr1p->vScrollHdl))->ctrlOwner;
  vLineSize=GetCRefCon(scr1p->vScrollHdl);
}
if(scr1p->hScrollHdl!=NIL)
{ theWindow=(*(scr1p->hScrollHdl))->ctrlOwner;
  hLineSize=-GetCRefCon(scr1p->hScrollHdl);
}
if((vLineSize==0)&&(hLineSize==0)) return;

ScrollSectRect(scr1p,theWindow,&dragRect);
GlobalToLocal(&where);
if(!PtInRect(where,&dragRect)) return;
lastWhere=where;
while(StillDown())
{ GetMouse(&where);
  if(PtInRect(where,&dragRect))
  { if(vLineSize>0) /* vert scrolling */
    { if(vAmt=(lastWhere.v-where.v)/vLineSize)
      { SetSelectScroll(scr1p->vScrollHdl);
        SelectScroll(vAmt);
        lastWhere.v-=vAmt*vLineSize;
      }
    }
    if(hLineSize>0) /* horiz scrolling */
    { if(hAmt=(lastWhere.h-where.h)/hLineSize)
      { SetSelectScroll(scr1p->hScrollHdl);
        SelectScroll(hAmt);
        lastWhere.h-=hAmt*hLineSize;
      }
    }
  }
}

/*Do joystick scrolling as long as mouse is down
1. where: mouse location in Global coordinates
Note: assumes scrolling in current active window */

JoyStickScroll(where)
Point where;
{
  int vAmt,hAmt;
  Point newWhere;

  if((scr1p->vScrollHdl!=NIL)&&(scr1p->hScrollHdl!=NIL))
    return;
  GlobalToLocal(&where);
  while(StillDown())
  { GetMouse(&newWhere);
    vAmt=(newWhere.v-where.v)/vJoySpeed;
    if((scr1p->vScrollHdl!=NIL)&&vAmt)
    { SetSelectScroll(scr1p->vScrollHdl);
      SelectScroll(vAmt);
    }
    hAmt=(newWhere.h-where.h)/hJoySpeed;
    if((scr1p->hScrollHdl!=NIL)&&hAmt)
    { SetSelectScroll(scr1p->hScrollHdl);
      SelectScroll(hAmt);
    }
  }
}

/*Set joy stick speed in pixels per speed increment
1. v or hSpeed: vertical or horizontal settings */

SetJoySpeed(vSpeed,hSpeed)
int vSpeed,hSpeed;
{

```

```

  vJoySpeed=vSpeed;
  hJoySpeed=hSpeed;
}

/*If mouse is outside content region, scroll 1 line
towards mouse location and return TRUE,
otherwise just return FALSE
Note: assumes scrolling in current active window */

FollowMouse()
{
  Point where;
  RgnHandle tempRgn;
  int followed=FALSE;

  GetMouse(&where);
  GetClip(tempRgn=NewRgn());
  ClipRect(&thePort->portRect);

  if(scr1p->hScrollHdl!=NIL) /* horiz tracking */
  { if(where.h<scr1p->horizScrollRect.left)
    { SetSelectScroll(scr1p->hScrollHdl);
      SelectScroll(-1);
      followed=TRUE;
    }
    else if(where.h>scr1p->horizScrollRect.right)
    { SetSelectScroll(scr1p->hScrollHdl);
      SelectScroll(1);
      followed=TRUE;
    }
  }
  if(scr1p->vScrollHdl!=NIL) /* vert tracking */
  { if(where.v<scr1p->vertScrollRect.top)
    { SetSelectScroll(scr1p->vScrollHdl);
      SelectScroll(-1);
      followed=TRUE;
    }
    else if(where.v>scr1p->vertScrollRect.bottom)
    { SetSelectScroll(scr1p->vScrollHdl);
      SelectScroll(1);
      followed=TRUE;
    }
  }

  SetClip(tempRgn);
  DisposeRgn(tempRgn);
  return(followed);
}

/*Handle automatic scrolling of Text Edit Windows
Note: assumes scrolling in current active window */

pascal Boolean SMClikLoop()
{
  Point where;
  RgnHandle tempRgn;

  GetMouse(&where);
  if(PtInRect(where,&(**scr1p->hTE).viewRect))
    return(TRUE);
  GetClip(tempRgn=NewRgn());
  ClipRect(&thePort->portRect);

  if(scr1p->hScrollHdl!=NIL) /* horiz tracking */
  { if(where.h<(**scr1p->hTE).viewRect.left)
    { SetSelectScroll(scr1p->hScrollHdl);
      SelectScroll(-1);
    }
    else if(where.h>(**scr1p->hTE).viewRect.right)
    { SetSelectScroll(scr1p->hScrollHdl);
      SelectScroll(1);
    }
  }
  if(scr1p->vScrollHdl!=NIL) /* vert tracking */

```

```

    ( if(where.v<(**scrip->hTE).viewRect.top)
      ( SetSelectScroll(scrip->vScrollHdl);
        SelectScroll(-1);
      )
    else if(where.v<(**scrip->hTE).viewRect.bottom)
      ( SetSelectScroll(scrip->vScrollHdl);
        SelectScroll(1);
      )
    )

    SetClip(tmpRgn); /* must restore clip region */
    DisposeRgn(tmpRgn);
    return(TRUE);
  }

  /*Scroll window from oldValue to newValue
   - internal use */

  ThumbMove(theWindow,oldValue,newValue)
  WindowPtr theWindow;
  int oldValue,newValue;
  {
    int units=oldValue-newValue;

    if(units)
      ScrollWindow(theWindow,units);
  }

  /*Do the actual scrolling. When required this routine
  calls UpdateWindow() which must be supplied by the
  application - internal use */

  ScrollWindow(theWindow,units)
  WindowPtr theWindow;
  int units;
  {
    int absunits,rectLength;
    Boolean hasRuler=FALSE;
    RgnHandle tmpRgn;
    GrafPtr saveport;
    Rect rulerRect;

    GetPort(&saveport);
    SetPort(theWindow);
    absunits= units<0 ? -units : units;

    /* scroll text windows*/
    if(scrip->hTE!=NIL)
    { if(scrollType==HorizBar)
      { TEScroll(rectLength=lineSize*units,0,
        scrip->hTE);
        if(scrip->vRectTopLeft.v!=
          scrip->hRectTopLeft.v)
        { hasRuler=TRUE;
          rulerRect.left=scrip->hRectTopLeft.h;
          rulerRect.top=scrip->hRectTopLeft.v;
          rulerRect.bottom=scrip->vRectTopLeft.v;
          rulerRect.right=
            scrip->horizScrollRect.right;
        }
      }
    else
    { TEScroll(0,rectLength=lineSize*units,
      scrip->hTE);
      if(scrip->vRectTopLeft.h!=
        scrip->hRectTopLeft.h)
      { hasRuler=TRUE;
        rulerRect.left=scrip->vRectTopLeft.h;
        rulerRect.top=scrip->vRectTopLeft.v;
        rulerRect.right=scrip->hRectTopLeft.h;
        rulerRect.bottom=
          scrip->vertScrollRect.bottom;
      }
    }
  }
}

```

```

  /*if rulers in text windows, scroll them*/
  if(hasRuler)
  { tmpRgn=NewRgn();
    if(scrollType==HorizBar)
      ScrollRect(&rulerRect,lineSize*units,0,
        tmpRgn);
    else
      ScrollRect(&rulerRect,0,lineSize*units,
        tmpRgn);
    InvalRgn(tmpRgn);
    DisposeRgn(tmpRgn);
  }
}

/*Scrolling more than 1 page in graphic windows */
else if(absunits>linesVis)
{ if(scrollType==HorizBar)
  { InvalRect(&scrip->horizScrollRect);
    else
      InvalRect(&scrip->vertScrollRect);
  }

  /*Fractional page scrolling in graphic windows */
  else
  { tmpRgn=NewRgn();
    if(scrollType==HorizBar)
      ScrollRect(&scrip->horizScrollRect,
        lineSize*units,0,tmpRgn);
    else
      ScrollRect(&scrip->vertScrollRect,
        0,lineSize*units,tmpRgn);
    InvalRgn(tmpRgn);
    DisposeRgn(tmpRgn);
  }

  /*If required, call Update process */
  if((scrip->hTE==NIL)||hasRuler)
    UpdateWindow(theWindow,scrollType);

  SetPort(saveport);
}

/*Scroll the line and column numbers given by selLine
and selColumn into view
Note: assumes scrolling in current active window */

FixInsertPt(selLine,selColumn)
int selLine,selColumn;
{
  if(scrip->vScrollHdl!=NIL)
    MoveInsertPt(selLine,VertBar);
  if(scrip->hScrollHdl!=NIL)
    MoveInsertPt(selColumn,HorizBar);
}

/*Scroll begining of selection of Text Edit window into
view
Note: assumes scrolling in current active window */

FixTEInsertPt()
{
  int selLine,selColumn=0,textLength,*line0;
  WindowPtr TEWindow;
  GrafPtr saveGraf;
  register int thePt,jumpSize,lineLook;

  if(scrip->hTE==NIL) return;
  if(scrip->vScrollHdl!=NIL)
    TEWindow=(*(scrip->vScrollHdl)->contr1owner;
  else if(scrip->hScrollHdl!=NIL)
    TEWindow=(*(scrip->hScrollHdl)->contr1owner;
  else
    return;
}

```

```

/*Binary search for current line */
thePt=(&scrIp->hTE).selStart; /*Set up pointers*/
lineLook=(&scrIp->hTE).nLines;
line0=&(&scrIp->hTE).lineStarts[0];
if(thePt==(&line0+lineLook)) /*Check last line*/
    selLine=lineLook-1;
else
( jumpSize=lineLook>>1; /*div by 2, but not 0*/
  lineLook-=jumpSize;
  while(jumpSize=jumpSize>>1){/*reduce jumpsize*/
    ( if(thePt<(&line0+lineLook))
      lineLook-=jumpSize; /*look back*/
      else if(thePt>(&line0+lineLook+1))
        lineLook+=jumpSize; /*look forward*/
      else
        ( selLine=lineLook; /*found line*/
          break;
        )
      )
    )
  )

/*Find column by getting width of text from start of
line to insertion point.*/
if(scrIp->hScrollHdl!=NIL)
( GetPort(&saveGraf);
  SetPort(TEWindow);
  HLock(&scrIp->hTE).hText);
  textLength=thePt-
    (&scrIp->hTE).lineStarts[selLine];
  selColumn=TextWidth(&(&scrIp->hTE).hText,
    (&scrIp->hTE).lineStarts[selLine],
    textLength);
  HUnlock(&scrIp->hTE).hText);
  selColumn/=(GetCRefCon(scrIp->hScrollHdl));
  SetPort(saveGraf);
)

/* Correct selLine in case scrolling line is smaller
than text lineHeight */
if(GetCRefCon(scrIp->vScrollHdl)<
  (&scrIp->hTE).lineHeight)
( selLine=selLine*(&scrIp->hTE).lineHeight/
  GetCRefCon(scrIp->vScrollHdl);
  if(selLine>GetCtlValue(scrIp->vScrollHdl))
    selLine+=1+(&scrIp->hTE).lineHeight/
      GetCRefCon(scrIp->vScrollHdl);
  )
  FixInsertPt(selLine,selColumn);
)

/*Scroll one direction to bring point into view
- internal use */

MoveInsertPt(selStart,barType)
int selStart,barType;
(
  int curValue,theLinesVis,units;
  ControlHandle scrollHdl;

  if(barType==VertBar)
  ( theLinesVis=scrIp->vertLinesVis;
    scrollHdl=scrIp->vScrollHdl;
  )
  else
  ( theLinesVis=scrIp->horizLinesVis;
    scrollHdl=scrIp->hScrollHdl;
  )

  curValue=GetCtlValue(scrollHdl);
  if(selStart<curValue)
    units=selStart-curValue;
  else if(selStart>curValue+theLinesVis-1)
    units=selStart-curValue-theLinesVis+1;
  else

```

```

    return;

    SetSelectScroll(scrollHdl);
    SelectScroll(units);
  )

/*Call TEClick - call from here insures SMClikLoop Code
available. Meaning of parameters identical to
Toolbox routine TEClick() */

ScrollTEClick(where,theShift,hTE)
Point where;
Boolean theShift;
TEHandle hTE;
(
  TEClick(where,theShift,hTE);
)

/*If ScrollInfo in heap, dispose of pointer when done.
Do not call if ScrollInfo not in heap*/

DisposeScroll(theScrIp)
ScrollPtr theScrIp;
(
  DisposPtr(theScrIp);
)

/*Return the refCon in ScrollInfo record */

long GetSRefCon(theScrIp)
ScrollPtr theScrIp;
(
  return(theScrIp->refCon);
)

/*Set the refCon in ScrollInfo record */

SetSRefCon(theScrIp,sRefCon)
ScrollPtr theScrIp;
long sRefCon;
(
  theScrIp->refCon=sRefCon;
)

/*Return the current active ScrollPtr */

ScrollPtr ActiveScrollPtr()
(
  return(scrIp);
)

```

Listing: ScrollerMain.c

/*Application Scroller, version 1.0 (August, 26, 1988)

This application demonstrate features and use of the
Scrolling manager

© 1988 John A. Nairn, All Rights Reserved */

```

/*Includes from Lightspeed C */
#include <QuickDraw.h>
#include <MacTypes.h>
#include <FontMgr.h>
#include <WindowMgr.h>
#include <MenuMgr.h>
#include <TextEdit.h>
#include <EventMgr.h>
#include <DeskMgr.h>
#include <DialogMgr.h>
#include <ToolboxUtil.h>
#include <ControlMgr.h>
#include "ScrollMgr.h" /*Scrolling Manager header*/

```

```

enum ( apple_menu=255,file_menu,edit_menu );
enum ( drawCursor=300,handCursor,joyCursor );
#define sample_text 500
#define about_dlog 998
#define NIL 0L /* Some constants */
#define RULERWIDTH 15
#define TOOLWIDTH 20
#define about_item 1 /* Menu items */
enum ( new_window=1,graphic_window=3,text_edit_window,
      vert_bar=6,horiz_bar,vert_ruler=9,horiz_ruler,
      vert_tool_bar,horiz_tool_bar,quit=14 );

WindowPtr myWindow=NIL; /* Global Variables */
TEHandle myHTE=NIL;
ScrollInfo scrollRecord;
ScrollPtr myScrlp;
Point polygon[200];
Rect dragrect;
int window_type=graphic_window,has_vBar=TRUE;
int has_vRuler=FALSE,has_vTools=FALSE,has_hRuler=FALSE;
int has_hTools=FALSE,has_hBar=TRUE;
int vRuler,vTools,hRuler,hTools,polySize=0;

/* Main Program - Initialize and start event loop */

main()
(
    MenuHandle menu;
    Rect r;

    InitGraf(&thePort);
    InitFonts();
    InitWindows();
    InitCursor();
    FlushEvents(everyEvent,0);
    InitMenus();
    TEInit();
    InitDialogs(0L);
    r=screenBits.bounds; /* Window dragging Rect */
    SetRect(&dragrect,4,24,r.right-4,r.bottom-4);
    menu=GetMenu(apple_menu); /*DA menu*/
    AddResMenu(menu,'DRVR');
    InsertMenu(menu,0);
    menu=GetMenu(file_menu); /*File menu*/
    InsertMenu(menu,0);
    CheckItem(menu>window_type,TRUE);
    CheckItem(menu>horiz_bar,has_vBar);
    CheckItem(menu>vert_bar,has_hBar);
    InsertMenu(GetMenu(edit_menu),0); /*Edit Menu*/
    MaintainMenus();
    do_about();
    event_loop();
)

/*Enable or Disable Edit Menu */

MaintainMenus()
(
    if(FrontWindow()==myWindow)
    ( if(myHTE==NIL)
        DisableItem(GetMHandle(edit_menu),0);
      else
        EnableItem(GetMHandle(edit_menu),0);
    )
    else
        EnableItem(GetMHandle(edit_menu),0);
    DrawMenuBar();
)

/* Maintain cursor according to window type, keys
down, and mouse location */

MaintainCursor()
(

```

```

KeyMap theKeys;
RgnHandle drawRgn;
Point where;

if(FrontWindow()!=myWindow)
    return;
else if(myWindow==NIL)
    InitCursor();
else
( drawRgn=ScrollSectRgn(myScrlp,myWindow);
  GetMouse(&where);
  if(!PtInRgn(where,drawRgn))
    InitCursor();
  else
  ( GetKeys(&theKeys);
    if((theKeys.Key[1]&4)&&(theKeys.Key[1]&1))
      SetCursor(*GetCursor(joyCursor));
    else if(theKeys.Key[1]&4)
      SetCursor(*GetCursor(handCursor));
    else if(myHTE!=NIL)
      SetCursor(*GetCursor(iBeamCursor));
    else
      SetCursor(*GetCursor(drawCursor));
  )
  DisposeRgn(drawRgn);
)

/*Program main event loop */

event_loop()
(
    EventRecord event;
    Boolean valid;
    char theChar;

    while (1)
    ( SystemTask();
      MaintainCursor();
      if(myHTE!=NIL) TEIdle(myHTE);
      valid=GetNextEvent(everyEvent,&event);
      if (!valid) continue;
      switch(event.what)
      ( case mouseDown:
          do_mouse_down(&event);
          break;
        case keyDown:
        case autoKey:
          theChar=event.message&charCodeMask;
          if(event.modifiers&cmdKey)
          ( if(event.what!=autoKey)
              do_menu(MenuKey(theChar));
            )
          else
          ( if(myHTE!=NIL)
              ( if(**myHTE).teLength<32760
                  ( TEKey(theChar,myHTE);
                    if(myScrlp->vScrollHd!=NIL)
                      SetScroll(myScrlp,
                        (**myHTE).nLines,VertBar);
                    FixTEInsertPt();
                  )
                else
                  SysBeep(10); /* TE full */
              )
            else
              SysBeep(10); /* bad key down */
          )
          break;
        case updateEvt:
          do_update(&event);
          break;
        case activateEvt:
          do_activate((WindowPtr)event.message,

```

```

        event.modifiers&activeFlag);
        break;
    default:
        break;
    }
}

/* Handle mouse down events */

do_mouse_down(eventp)
    EventRecord *eventp;
{
    WindowPtr windowp;
    int place_type=FindWindow(eventp->where,&windowp);
    Boolean theshift;
    KeyMap theKeys;

    switch(place_type)
    {
        case inMenuBar:
            do_menu(MenuSelect(eventp->where));
            break;
        case inSysWindow:
            SystemClick(eventp,windowp);
            MaintainMenus();
            break;
        case inContent:
            if(windowp!=FrontWindow())
            {
                SelectWindow(windowp);
                MaintainMenus();
            }
            else if(windowp==myWindow)
            {
                if(DoScroll(windowp,eventp->where)==FALSE)
                {
                    GetKeys(&theKeys);
                    if((theKeys.Key[1]&4)&&
                       (theKeys.Key[1]&1))
                        JoyStickScroll(eventp->where);
                    else if(theKeys.Key[1]&4)
                        DragScroll(eventp->where);
                    else if(myhTE==NIL)
                        TrackPoly(eventp->where);
                    else
                    {
                        GlobalToLocal(&eventp->where);
                        if(PtInRect(eventp->where,
                                   &(**myhTE).viewRect))
                        {
                            if(eventp->modifiers&shiftKey)
                                theshift=TRUE;
                            else
                                theshift=FALSE;
                            ScrollTEClick(eventp->where,
                                           theshift,myhTE);
                        }
                        else
                            SysBeep(10);
                    }
                }
            }
            break;
        case inDrag:
            DragWindow(windowp,eventp->where,&dragrect);
            break;
        case inGrow:
            grow_window(windowp,eventp->where);
            break;
        case inGoAway:
            if(TrackGoAway(windowp,eventp->where))
            {
                do_close(windowp);
                MaintainMenus();
            }
            break;
        default:
            break;
    }
}

```

```

/* Track mouse and draw a polygon - for Graphic windows*/

TrackPoly(where)
    Point where;
{
    int lm,tm;
    Point lastWhere;
    Rect drawRect;
    RgnHandle drawRgn;

    GlobalToLocal(&where);
    ScrollSectRect(myScrip,myWindow,&drawRect);
    if(!PtInRect(where,&drawRect))
        { SysBeep(10); /* illegal click location */
          return;
        }
    GetMargins(myScrip,&lm,inPixels,&tm,inPixels);
    if(polySize==0)
        { polygon[polySize].h=where.h+lm;
          polygon[polySize].v=where.v+tm;
        }
    else if(polySize==200)
        { SysBeep(10);
          return;
        }
    drawRgn=ScrollSectRgn(myScrip,myWindow);
    SetClip(drawRgn);
    PenMode(patXor);
    MoveTo(polygon[polySize].h-lm,polygon[polySize].v-tm);
    LineTo(where.h,where.v);
    lastWhere=where;
    while(StillDown())
        { GetMouse(&where);
          if(PtInRgn(where,drawRgn))
          {
              if((where.h!=lastWhere.h)||
                 (where.v!=lastWhere.v))
              {
                  MoveTo(polygon[polySize].h-lm,
                        polygon[polySize].v-tm);
                  LineTo(lastWhere.h,lastWhere.v);
                  MoveTo(polygon[polySize].h-lm,
                        polygon[polySize].v-tm);
                  LineTo(where.h,where.v);
                  lastWhere=where;
              }
          }
        }
    else
        {
            MoveTo(polygon[polySize].h-lm,
                  polygon[polySize].v-tm);
            LineTo(lastWhere.h,lastWhere.v);
            FollowMouse();
            GetMargins(myScrip,&lm,inPixels,&tm,inPixels);
            MoveTo(polygon[polySize].h-lm,
                  polygon[polySize].v-tm);
            LineTo(where.h,where.v);
            lastWhere=where;
        }
    }
    MoveTo(polygon[polySize].h-lm,polygon[polySize].v-tm);
    LineTo(lastWhere.h,lastWhere.v);
    PenMode(patCopy);
    if(lastWhere.h>drawRect.right)
        lastWhere.h=drawRect.right;
    if(lastWhere.h<drawRect.left)
        lastWhere.h=drawRect.left;
    if(lastWhere.v>drawRect.bottom)
        lastWhere.v=drawRect.bottom;
    if(lastWhere.v<drawRect.top)
        lastWhere.v=drawRect.top;
    MoveTo(polygon[polySize].h-lm,polygon[polySize].v-tm);
    LineTo(lastWhere.h,lastWhere.v);
    polygon[++polySize].h=lastWhere.h+lm;
    polygon[polySize].v=lastWhere.v+tm;
    ClipRect(&myWindow->portRect);
}

```

```

    DisposeRgn(drawRgn);
}

/* Close the scrolling demonstration window */
do_close(windowp)
WindowPtr windowp;
{
    if (myWindow!=NIL)
    { if(myhTE!=NIL)
      { ZeroScrap();
        TEToScrap();
        TEDispose(myhTE);
      }
      CloseWindow(windowp);
      myWindow=NIL;
      myhTE=NIL;
    }
}

/* Handle menu command events */
do_menu(command)
Long command;
{
    int menu_id=HiWord(command);
    int item=LoWord(command);
    if(menu_id!=0)
        do_menu_item(menu_id,item);
}

/* Do a menu command */
do_menu_item(menu_id,item)
int menu_id,item;
{
    char item_name[32];
    int newCheck;

    switch(menu_id)
    { case apple_menu:
      switch(item)
      { case about_item:
        do_about();
        break;
        default:
            GetItem(GetMHandle(menu_id),
                    item,item_name);
            OpenDeskAcc(item_name);
            HiliteMenu(0);
            MaintainMenus();
            break;
      }
      break;
      case file_menu:
      switch (item)
      { case new_window:
        do_close(myWindow);
        myNewWindow();
        HiliteMenu(0);
        MaintainMenus();
        break;
        case graphic_window:
        case text_edit_window:
            CheckItem(GetMHandle(file_menu),
                      window_type,FALSE);
            window_type=item;
            CheckItem(GetMHandle(file_menu),
                      window_type,TRUE);
            break;
        case vert_bar:
            newCheck=has_vBar=1-has_vBar;
            break;
        case horiz_bar:

```

```

            newCheck=has_hBar=1-has_hBar;
            break;
        case vert_ruler:
            newCheck=has_vRuler=1-has_vRuler;
            break;
        case horiz_ruler:
            newCheck=has_hRuler=1-has_hRuler;
            break;
        case vert_tool_bar:
            newCheck=has_vTools=1-has_vTools;
            break;
        case horiz_tool_bar:
            newCheck=has_hTools=1-has_hTools;
            break;
        case quit:
            do_close(myWindow);
            finish();
        default:
            break;
    }
    if(item==vert_bar)
        CheckItem(GetMHandle(file_menu),
                  item,newCheck);
    break;
    case edit_menu:
    if(!SystemEdit(item-1))
    { switch(item-1)
      { case undoCmd:
        break;
        case cutCmd:
            TECut(myhTE);
            break;
        case copyCmd:
            TECopy(myhTE);
            break;
        case pasteCmd:
            if(((long)(**myhTE).teLength+
              (long)TEGetScrapLen())<32768)
                TEPaste(myhTE);
            else
                SysBeep(10);
            break;
        case clearCmd:
            TEDelete(myhTE);
            break;
        default:
            break;
      }
    }
    /* reset scroll bar */
    if(myScrlp->vScrollHdl!=NIL)
        SetScroll(myScrlp,(**myhTE).nLines,
                  VertBar);

    /* if needed, move insertion point */
    if(((item-1)==cutCmd)||
        ((item-1)==clearCmd)||
        ((item-1)==pasteCmd))
        FixTEInsertPt();
    }
    default:
        break;
    }
    HiliteMenu(0);
}

/* Handle Update events */
do_update(event)
EventRecord *event;
{
    WindowPtr up_wind=(WindowPtr)event->message;
    UpdateWindow(up_wind,FALSE);
}

```



```

/* Update the window */
UpdateWindow(up_wind,scrollType)
WindowPtr up_wind;
int scrollType;
{
    GrafPtr save_graf;

    if(up_wind==myWindow)
    { GetPort(&save_graf);
      SetPort(up_wind);
      BeginUpdate(up_wind);
      ClipRect(&up_wind->portRect);
      EraseRect(&up_wind->portRect);
      if(!scrollType)
      { DrawControls(up_wind);
        ScrollDrawGrowIcon(myScrlp,up_wind);
      }
      draw_content(up_wind,scrollType);
      EndUpdate(up_wind);
      SetPort(save_graf);
    }
}

/*Activate the window */
do_activate(act_wind,isactivate)
WindowPtr act_wind;
int isactivate;
{
    Rect r;

    SetPort(act_wind);
    if(act_wind==myWindow)
    { ScrollDrawGrowIcon(myScrlp,act_wind);
      if(isactivate)
      { ScrollActivate(myScrlp);
        if(myhTE!=NIL)
        { TEActivate(myhTE);
          TEFromScrap();
        }
      }
      else
      { ScrollDeactivate(myScrlp);
        if(myhTE!=NIL)
        { TEDeactivate(myhTE);
          ZeroScrap();
          TEToScrap();
        }
      }
    }
}

/* handle window growing */
grow_window(windowp,mouse_point)
WindowPtr windowp;
Point mouse_point;
{
    long new_bounds;

    InvalBars(windowp); /*invalidate scroll bars*/
    if((new_bounds=GrowWindow(windowp,mouse_point,
                              &dragrect))==0)
        return;
    SizeWindow(windowp,LoWord(new_bounds),
               HiWord(new_bounds),(Boolean)TRUE);
    InvalBars(windowp); /*invalidate new scroll bars*/
    GrowScroll(windowp);/*update scroll bar window*/
}

/* Open new scrolling example window */

```

```

myNewWindow()
{
    Rect r,wrect;
    int hLineSize=0,vLineSize=0,vTotalLines=0;
    int vToolSize=0,vRulerSize=0,hToolSize=0,hRulerSize=0;
    int vScrollTop=0,hScrollLeft=0,hTotalLines=0;
    Boolean includeText=FALSE;
    FontInfo theFont;
    Handle textHdl;

    r=screenBits.bounds;
    SetRect(&wrect,20,60,r.right-20,r.bottom-20);
    myWindow=NewWindow(NIL,&wrect,"pScrolling Example",
                      1,documentProc,-1L,1,0L);

    polySize=0;
    SetPort(myWindow);
    TextFont(geneva);
    TextSize(12);
    /*Note: add 1 to ruler and tool widths for line*/
    if(has_vRuler) vRulerSize=RULERWIDTH+1;
    if(has_hRuler) hRulerSize=RULERWIDTH+1;
    if(has_vTools)
    { vToolSize=TOOLWIDTH+1;
      hScrollLeft=TOOLWIDTH+1;
    }
    if(has_hTools)
    { hToolSize=TOOLWIDTH+1;
      vScrollTop=TOOLWIDTH+1;
    }
    if(has_hBar) hLineSize=18;
    if(window_type==graphic_window)
    { vTotalLines=hTotalLines+40;
      if(has_vBar) vLineSize=18;
    }
    else
    { if(has_vBar)
      { GetFontInfo(&theFont);
        vLineSize=theFont.ascent+theFont.descent+
                  theFont.leading;
      }
      vTotalLines=0;
      if(has_hBar)
        hTotalLines=(wrect.right-wrect.left-vToolSize-
                     vRulerSize-SBarWidth)/hLineSize;
      includeText=TRUE;
    }

    /* Set up scroll bars */
    myScrlp=SetScrollWindow(myWindow,&scrollRecord,
                           includeText,vLineSize,vTotalLines,vToolSize,
                           vRulerSize,vScrollTop,hLineSize,hTotalLines,
                           hToolSize,hRulerSize,hScrollLeft,NIL);

    /*If text window, add some sample text */
    myhTE=myScrlp->hTE;
    if(myhTE!=NIL)
    { textHdl=GetResource('GNRL',sample_text);
      HLock(textHdl);
      TEText(*textHdl,SizeResource(textHdl),myhTE);
      HUnlock(textHdl);
      ReleaseResource(textHdl);
      if(myScrlp->vScrollHdl!=NIL)
        SetScroll(myScrlp,(*myhTE).nLines,VertBar);
    }

    /* Set current window tool and ruler flags*/
    vTools=has_vTools;
    hTools=has_hTools;
    vRuler=has_vRuler;
    hRuler=has_hRuler;
}

/* Draw the window contents */

```

```

draw_content(windowp,scrollType)
WindowPtr windowp;
int scrollType;
{
    register int i;
    int lineSize=20,lm,tm,tick,nexttick;
    int vtoolsize=0,htoolsize=0,xtick,ytick,vtotal,htotal;
    static char vertTools[11]={'0','V','E','R','T',' ',' ','T',
        '0','0','L','S'};
    static char horizTools[12]={'H','0','R','I','Z',
        ' ','T','0','0','L','S'};
    static int length[8]={0,5,8,5,11,5,8,5};
    char nstr[20];
    Rect drawRect,tempRect;
    RgnHandle drawRgn;

    GetMargins(myScrlp,&lm,inPixels,&tm,inPixels);
    ClipRect(NonScrollRect(myScrlp>windowp,&drawRect));
    if(vTools) vtoolsize=TOOLWIDTH;
    if(hTools) htoolsize=TOOLWIDTH;

    if(vTools&&(!scrollType)) /*vert tool bar*/
    { MoveTo(vtoolsize,drawRect.top);
      LineTo(vtoolsize,drawRect.bottom);
      TextFace(outline+bold+shadow);
      for(i=1;i<=vertTools[0];i++)
      { MoveTo(0,htoolsize+20*i);
        LineTo(vtoolsize,htoolsize+20*i);
        MoveTo(6,htoolsize+20*i-5);
        DrawChar(vertTools[i]);
      }
      TextFace(0);
    }

    if(hTools&&(!scrollType)) /*horiz tool bar*/
    { MoveTo(drawRect.left,htoolsize);
      LineTo(drawRect.right,htoolsize);
      TextFace(outline+bold+shadow);
      for(i=1;i<=horizTools[0];i++)
      { MoveTo(vtoolsize+25*i,0);
        LineTo(vtoolsize+25*i,htoolsize);
        MoveTo(vtoolsize+25*i-17,15);
        DrawChar(horizTools[i]);
      }
      TextFace(0);
    }

    if(vTools&&hTools&&(!scrollType))
        markX(0,0,htoolsize,vtoolsize);

    if(vRuler&&hRuler&&(!scrollType))
        markX(vtoolsize,htoolsize,RULERWIDTH,RULERWIDTH);

    TextSize(9);
    if(vRuler&&(scrollType!=HorizBar))
    { vtotal=vtoolsize+RULERWIDTH;
      MoveTo(vtotal,drawRect.top+htoolsize);
      LineTo(vtotal,drawRect.bottom);
      drawRect.top=htoolsize+RULERWIDTH*hRuler;
      ClipRect(&drawRect);
      tick=tm/72;
      nexttick=drawRect.top+72*tick-tm-72;
      while((nexttick+=72)<=drawRect.bottom)
      { MoveTo(vtoolsize,nexttick);
        LineTo(vtotal,nexttick);
        if(tick)
        { NumToString((long)tick,nstr);
          MoveTo(vtotal-StringWidth(nstr),
              nexttick+8);
          DrawString(nstr);
        }
        for(i=1;i<=7;i++)
        { MoveTo(vtotal,ytick=nexttick+9*i);

```

```

          LineTo(vtotal-length[i],ytick);
        }
        tick++;
      }
    }

    if(hRuler&&(scrollType!=VertBar))
    { htotal=htoolsize+RULERWIDTH;
      drawRect.top=0;
      ClipRect(&drawRect);
      MoveTo(drawRect.left+vtoolsize,htotal);
      LineTo(drawRect.right,htotal);
      drawRect.left=vtoolsize+RULERWIDTH*vRuler;
      ClipRect(&drawRect);
      tick=lm/72;
      nexttick=drawRect.left+72*tick-lm-72;
      while((nexttick+=72)<=drawRect.right+8)
      { MoveTo(nexttick,htoolsize);
        LineTo(nexttick,htotal);
        if(tick)
        { NumToString((long)tick,nstr);
          MoveTo(nexttick-1-StringWidth(nstr),
              htoolsize+10);
          DrawString(nstr);
        }
        for(i=1;i<=7;i++)
        { MoveTo(xtick=nexttick+9*i,htotal);
          LineTo(xtick,htotal-length[i]);
        }
        tick++;
      }
    }
    TextSize(12);

```

```

    if(myhTE==NIL) /*Graphic Window Update*/
    { drawRgn=ScrollSectRgn(myScrlp>windowp);
      SetClip(drawRgn);
      MoveTo(polygon[0].h-lm,polygon[0].v-tm);
      for(i=1;i<=polySize;i++)
          LineTo(polygon[i].h-lm,polygon[i].v-tm);
      ClipRect(&windowp->portRect);
      DisposeRgn(drawRgn);
    }
    else /*Text Window Update*/
    { ClipRect(&windowp->portRect);
      tempRect=(**myhTE).viewRect;
      TEUpdate(&tempRect,myhTE);
    }
}

```

/*Draw X in window */

```

markX(beginx,beginy,xwidth,ywidth)
int beginx,beginy,xwidth,ywidth;
{ MoveTo(beginx,beginy);
  LineTo(beginx+xwidth,beginy+ywidth);
  MoveTo(beginx,beginy+ywidth);
  LineTo(beginx+xwidth,beginy);
}

```

/* Present about box and wait for mouse click */

```

do_about()
{
    DialogPtr aboutBox;
    EventRecord event;

    aboutBox=GetNewDialog(about_dlog,NIL,-1L);
    DrawDialog(aboutBox);

    /* wait for key or mouse click -
       from MacTutor, V4, #3, pg10 */
    while(!EventAvail(keyDownMask|autoKeyMask|mouseDownMask,
        &event))

```

```

SystemTask();

if(event.what==mouseDown) /* remove mouse down */
( GlobalToLocal(&event.where);
  if(PtInRect(event.where,&aboutBox->portRect))
    GetNextEvent(mDownMask,&event);
)

DisposDialog(aboutBox);
)

/* Exit to shell after saving TE scrap */

finish()
(
  ZeroScrap();
  TEToScrap();
  LoadScrap();
  ExitToShell();
)

```

Listing: Scroller Project.R

```

*****
*
* RMaker resource file source for *
* Application Scroller *
*
* © 1988 John A. Nairn *
*
*****

```

Scroller Project.rsrc
 ???????

```

Type SCRL = STR
,0
Demo of the Scrolling Manager - ++
August 13, 1988

```

```

Type FREF
,128
APPL 0

```

```

Type BNDL
,128
SCRL 0
ICN#
0 128
FREF
0 128

```

```

Type MENU
,255 (0) ;;Apple Menu
\14
About Scroller...
(-

```

```

,256 (0)
File ;;File Menu
New Window/N
(-
Graphics Window
Text Edit Window
(-
Vertical Scroll Bar
Horizontal Scroll Bar
(-
Vertical Ruler
Horizontal Ruler
Vertical Tool Bar
Horizontal Tool Bar
(-
Quit/Q

```

```

,257 (0)
Edit ;;Edit Menu
Undo/Z
(-
Cut/X
Copy/C
Paste/V
Clear

Type DITL ;;Text for About Box
,998
3 ;;Number of items

```

```

StatText Disabled
11 58 29 320
Scroller (© 1988, All Rights Reserved)

```

```

StatText Disabled
35 10 88 372
This application demonstrates the ++
features of the Scrolling Manager ++
library of routines which can help ++
other applications easily accomplish ++
window scrolling.

```

```

StatText Disabled
93 87 175 288
For information contact:\0D ++
John A. Nairn\0D ++
7108 S. Pine Cone St.\0D ++
Salt Lake City, UT 84121\0D ++
(801)-942-7768

```

```

Type DLOG ;;About dialog
,998
About Box
62 62 250 445
Visible NoGoAway
3 ;;Process ID
0 ;;refCon
998 ;;Item list ID

```

```

Type GNRL ;;Text Edit window text
,500
.H
2020 2020 2054 6865 2074 6578 7420 6564
6974 696E 6720 696E 2074 6869 7320 7769
6E64 6F77 2069 7320 6861 6E64 6C65 6420
6279 2074 6865 2054 6578 7420 4564 6974
206D 616E 6167 6572 2E20 5468 6520 7363
726F 6C6C 696E 6720 6973 2068 616E 646C
6564 2062 7920 7468 6520 5363 726F 6C6C
696E 6720 4D61 6E61 6765 722E 0D0D 2020
2020 2059 6F75 2063 616E 2074 7970 6520
696E 2074 6578 742C 2061 6E64 2075 7365
2074 6865 2063 7574 2C20 636F 7079 2C20
7061 7374 652C 2061 6E64 2063 6C65 6172
2063 6F6D 6D61 6E64 732E 2057 6865 6E20
7365 6C65 6374 696E 6720 7465 7874 2066
6F72 2065 6469 7469 6E67 2061 6E64 2079
6F75 2064 7261 6720 6F75 7473 6964 6520
7468 6520 7769 6E64 6F77 2C20 7468 6520
5363 726F 6C6C 696E 6720 4D61 6E61 6765
7220 7769 6C6C 2061 7574 6F6D 6174 6963
616C 6C79 2073 6372 6F6C 6C20 7468 6520
7769 6E64 6F77 2074 6F20 666F 6C6C 6F77
2074 6865 206D 6F75 7365 2E20 5468 6520
5363 726F 6C6C 696E 6720 4D61 6E61 6765
7220 616C 736F 2069 6E73 7572 6573 2074
6861 7420 7468 6520 696E 7365 7274 696F
6E20 6261 7220 6973 2061 7574 6F6D 6174
6963 616C 6C79 2073 6372 6F6C 6C65 6420
696E 746F 2076 6965 7720 7768 656E 2061

```

```

7070 726F 7072 6961 7465 2E0D 0D20 2020
2020 486F 6C64 696E 6720 646F 776E 2074
6865 206F 7074 696F 6E20 6865 7920 696D
706C 656D 656E 7473 2064 7261 6720 7363
726F 6C6C 696E 672E 2048 6F6C 6469 6E67
2064 6F77 6E20 7468 6520 7368 6966 7420
616E 6420 6F70 7469 6F6E 206B 6579 7320
696D 706C 656D 656E 7473 206A 6F79 2073
7469 636B 2073 6372 6F6C 6C69 6E67 2E

```

```

Type ICN# = GNRL ;;Application Icon
,128
.H

```

```

FFFF FFFF 8000 0001 BC18 00A1 A5D3 6EAD
A40A 4AA1 BDD8 4EAD 8000 0001 FFFF FFFF
8000 0111 8000 0139 8A00 017D AA00 0139
AA00 01FF AE00 01AB B100 0155 A71F 81FF
B11F C183 A26F E183 BEE0 71FF BEC8 9129
BE95 5145 A388 9929 A302 0911 FFFF FFFF
86BE A101 9563 6971 AEA2 B55D C563 6375
AEA2 B525 9563 693D 86BE A101 FFFF FFFF
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF

```

```

FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF

```

```

TYPE CURS = GNRL

```

```

,300 ;;Pencil Cursor
.H

```

```

00F0 0088 0108 0190 0270 0220 0420 0440
0840 0880 1080 1100 1E00 1C00 1800 1000
00F0 00F8 01F8 01F0 03F0 03E0 07E0 07C0
0FC0 0F80 1F80 1F00 1E00 0000 0000 0000
0010 0003

```

```

,301 ;;Hand Cursor
.H

```

```

0180 1A70 2648 264A 124D 1249 6809 9801
8802 4002 2002 2004 1004 0808 0408 0408
0180 1BF0 3FF8 3FFA 1FFF 1FFF EFFF FFFF
FFFE 7FFE 3FFE 3FFC 1FFC 0FF8 07F8 07F8
0009 0008

```

```

,302 ;;JoyStick Cursor
.H

```

```

0080 01C0 03E0 0080 0080 1084 3086 7FFF
3086 1084 0080 0080 03E0 01C0 0080 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0007 0008

```



Programmer's Workshop

April Fool's INIT: Scrolling the Menu Bar!?



Mike Scanlin
San Diego, CA
Volume 5, Number 4

Probably everyone who reads this magazine knows at least one person who owns a Mac that they'd like to play a joke on. This INIT resource might be just the thing you're looking for. It is a patch to `_GetNextEvent` that will scroll the menu bar off the right side of the screen (leaving the menu bar blank) when the mouse is clicked in the very top pixel of the menu bar with an odd horizontal position. It does nothing if the mouse is clicked anywhere else on the screen, so it is possible to use the menus when this patch is installed, just so long as you don't click in the very top pixel at an odd position. The menus will come back eventually (after a few sarcastic messages are displayed in the empty menu bar), so this is a pretty harmless patch (although extremely annoying according to my guinea pig roommate). Command key equivalents for current menus (like `COMMAND-Q`) will work even though no menus are shown. It is possible to remove the patch once it is installed by typing `COMMAND-SHIFT-OPTION-TAB`. The patch will beep to let you know it has been removed.

HOW IT WORKS

This patch to `_GetNextEvent` is an application of the technique I used in the Shift Mod patch (September 1987 issue of *MacTutor*). It is a tail patch on `_GetNextEvent` that looks at the event being returned and intercepts it if it is a `MouseDown` event with a vertical position of zero and an odd horizontal position. When it finds such an event the menu bar is scrolled off the screen and then `_DrawMenuBar` and `_HiliteMenu` are patched to do nothing. The reason those two traps are patched is because we don't want anything drawn in the menu bar while we're displaying our messages. The menu bar is restored by restoring those two traps and then calling `_DrawMenuBar`. Note that because we are patching `_DrawMenuBar` the user will not be able to get the menus back by quitting the current application and/or launching another application (because every application uses `_DrawMenuBar` to put their menus up).

Once we have scrolled the menus off, the patch to `_GetNextEvent` will intercept any `MouseDown` event anywhere in the menu bar area and display a message in the menu bar for 1 second or until the mouse button is let go, whichever is longer. After four such messages, the patches to `_DrawMenuBar` and `_HiliteMenuBar` are removed and `_DrawMenuBar` is called to draw the current menu bar (which may not necessarily be the one that was scrolled off since the user could have changed applications in the mean time).

To do the actual scrolling you would expect me to use `_ScrollRect`. That's what I tried at first, but it was too slow (took about 10 seconds for the menu bar to get all the way off the screen). Then I tried scrolling by 3 or 4 pixels at a time, but it

wasn't smooth enough. The version you see here is scrolling by rotating bits in screen memory (via the 68000's `roxr` instruction). Although somewhat unorthodox (Apple wouldn't approve) it now only takes about 2 seconds to smoothly scroll the menu bar all the way off the screen.

PROBLEM

There is a potential problem with this patch. There is a danger in patching and unpatching traps that may be patched by other applications. For applications that patch traps this isn't normally a problem because they get a chance to restore the traps when they quit (before something else can patch them). But it is possible for our patches to be unpatched at a bad time. A "bad time" to unpatch would be after something else has patched on top of one of our patches. For instance, if we patch `_GetNextEvent` and then some other program patches it and then we unpatch ours, the second patch will be unpatched along with ours, but the application that made the second patch won't know about it. To partially correct this from happening, I have the `removePatch` routine check if something has patched `_GetNextEvent` after we have patched it (by calling `_GetTrapAddress` and comparing it to ourselves) and if it has, then ignore the request to unpatch ourselves. I did not do this for `_DrawMenuBar` and `_HiliteMenu`, but it is unlikely that other applications would patch either of those two (besides, this is only a joke — it's not something you're going to leave around in your System Folder forever).

One additional problem is that while this patch should work on any size monochrome monitor, it will not work on a Mac II color monitor. If anyone comes up with a simple fix, please send it in.

```
/* scrollMenuBarPatch.c      18 Nov 1987
 *
 * by Mike Scanlin and Andy Voelker
 *
 * This will install a patch to
 * _GetNextEvent that intercepts
 * mouseDown events if they occur in the
 * very top pixel-row of the screen. If
 * an event is intercepted, the menu bar
 * is scrolled off the right side of the
 * screen before the user can react.
 * After 4 more clicks in the menuBar
 * (with appropriately sarcastic messages
 * displayed after each one) the original
 * menuBar is restored. The patch can
 * be removed by typing
 * COMMAND-OPTION-SHIFT-TAB.
 */

#include "EventMgr.h"
#include "QuickDraw.h"
#include "asm.h"
```

```

/* low memory globals */
extern Handle MenuList : 0x0A1C;
extern Ptr ScrnBase : 0x0824;
extern GrafPtr WMgrPort : 0x09DE;

```

```

/* the traps we patch */
#define GetNextEventTrap 0xA970
#define DrawMenuBarTrap 0xA937
#define HiliteMenuTrap 0xA938

```

```

#define CLICKS_TO_MENU 4
#define MESSAGE_DELAY 60
#define MENU_BAR_HEIGHT 20
#define TAB_KEY 0x09
#define JMP_INSTRUCTION 0x4EF9

```

```

#define memFullErr -108
#define screenBits_bounds_right -110
#define screenBits_bounds_left -114

```

```
void main(void);
```

```
void main()
```

```
{
    asm {
```

```

/* This first section is the only part
 * that gets run initially. It gets some
 * space in the system heap and sets up a
 * patch to _GetNextEvent. The patch
 * won't do anything until the user
 * clicks in the very top pixel of the
 * menu bar (in which case it will scroll
 * the menu bar off the screen to the
 * right). */

```

```
    move.l D3, -(SP)
```

```

/* get the old trap address */
    move    #GetNextEventTrap, D0
    _GetTrapAddress

```

```

/* set the address for the JMP
 * instruction that calls the original
 * trap */
    lea     @origTrap, A1
    move.l  A0, (A1)

```

```

/* get some space in the system heap for
 * our patch */

```

```

    lea     @last, A0
    lea     @first, A1
    suba.l  A1, A0

```

```

/* D0 = length of patch */
    move.l  A0, D0

```

```
/* save length for _BlockMove */
```

```

    move.l  D0, D3
    _NewPtr SYS
    cmpi    #memFullErr, D0
    beq.s   @noPatch
    lea     @saveLoc, A1

```

```

/* save for removePatch */
    move.l  A0, (A1)

```

```

/* save for _BlockMove */
    move.l  A0, -(SP)

```

```

/* set the trap address to the space we
 * just got in the system heap. */
    move    #GetNextEventTrap, D0
    _SetTrapAddress

```

```
/* now move our patch into place */
```

```
    lea     @first, A0
```

```
/* (SP)+ is the result from _NewPtr */
```

```

    move.l  (SP)+, A1
    move.l  D3, D0
    _BlockMove

```

```

@noPatch
    move.l  (SP)+, D3

```

```

/* this is the end of the installation
 * part, but we can't do an RTS here
 * because LSC needs to clean up. So we
 * fall through to the end. */
    bra     @last

```

```

/*****
 * Here's the new _GetNextEvent. It calls
 * the existing _GetNextEvent and then
 * checks if a mouseDown or keyDown event
 * is being reported. If not, the event
 * is passed to the application
 * unmodified. If we end up using the
 * event ourselves, a null event is
 * returned to the application.
 *****/

```

```
@first
```

```

/* pop the original return address and
 * save it */

```

```

    lea     @exitAddress, A0
    move.l  (SP)+, (A0)

```

```

/* save ptr to event record so we can get
 * at it later */

```

```

    lea     @eventRecPtr, A0
    move.l  (SP), (A0)

```

```

/* set the return address to our patch */
    pea     @tailPatch

```

```

/* the nops get filled with the address
 * of the original _GetNextEvent */
    dc      JMP_INSTRUCTION

```

```
@origTrap
```

```

    nop
    nop

```

```

/* this is where it comes after the
 * normal _GetNextEvent processing */

```

```

@tailPatch
    movem.l A1/D0-D3, -(SP)

```

```

    lea     @eventRecPtr, A0
    move.l  (A0), A0

```

```

/* check if it's a keydown event that
 * says to remove ourselves. This is the
 * only keyDown that we intercept. */

```

```

    move
    OFFSET(EventRecord, what)(A0), D0
    cmpi    #keyDown, D0
    bne.s   @noKeyDown

```

```

/* the key to remove the patch is
 * COMMAND-SHIFT-OPTION-Tab */

```

```

    move.l
    OFFSET(EventRecord, message)(A0), D0
    cmpi.b  #TAB_KEY, D0
    bne.s   @noKeyDown

```

```

    move
    OFFSET(EventRecord, modifiers)(A0), D0
    andi
    #optionKey+cmdKey+shiftKey, D0
    eori
    #optionKey+cmdKey+shiftKey, D0
    beq.s   @removePatch

```

```
@noKeyDown
```

```
/* if it's not a mousedown event, then
```

```

* ignore it */
cmpi    #mouseDown,D0
bne.s   @patchExit

/* if we've already scrolled the menu
* list, then don't scroll it again */
lea     @menus,A1
tst     (A1)
bne.s   @alreadyGone

/* if no menus exist, then leave */
move.l   MenuList,A1
move.l   (A1),A1
tst     (A1)
beq.s   @patchExit

/* if the mouse is not at an odd
* location, then leave */
move.l   OFFSET(EventRecord,where)(A0),D0
andi     #1,D0
beq.s   @patchExit

/* if the mouse is not at the very top
* pixel of the menu bar, then leave */
move.l   OFFSET(EventRecord,where)(A0),D0
/* put vertical coordinate in low word */
swap     D0
cmpi     #1,D0
bge.s   @patchExit

/* now we're set to scroll that puppy. */
bsr      @scrollMenuBar

/* save the fact that the menu bar that
* was just scrolled */
lea     @menus,A0
move     #1,(A0)

/* patch _DrawMenuBar and _HiliteMenu to
* do nothing if and when they're
* called */
bsr      @disableTraps

lea     @clicks,A1
move     #CLICKS_TO_MENU,(A1)

bra.s    @returnNullEvent

@alreadyGone
/* if mouse is not in menu bar, then
* leave */
move.l   OFFSET(EventRecord,where)(A0),D0
/* put vertical coordinate in low word */
swap     D0
cmpi     #MENU_BAR_HEIGHT,D0
bge.s   @patchExit

/* print a message in the menu bar */
bsr      @drawAMessage

lea     @clicks,A0
subi     #1,(A0)
bne.s   @returnNullEvent

/* now that we're finished playing,
* restore _DrawMenuBar and
* _HiliteMenu */
bsr      @restoreMenus

@returnNullEvent
/* set the event to null */
lea     @eventRecPtr,A0

```

```

move.l   (A0),A0
clr
OFFSET(EventRecord,what)(A0)
/* change _GetNextEvent's return value to
* false. The 20 is for the 5 regs that
* are saved on the stack at this point
*/
clr      20(SP)

@patchExit
move.l   (SP)+,A1/D0-D3
/* JMP to the place that called
* _GetNextEvent */
dc       JMP_INSTRUCTION
@exitAddress
nop
nop

/* this is where it comes to remove the
* _GetNextEvent patch */
@removePatch
/* if the menus aren't shown, then
* restore them before leaving */
bsr      @restoreMenus

/* check if we are the most recent patch
* to _GetNextEvent. If we're not, then
* don't unpatch. */
move     #GetNextEventTrap,D0
_GetTrapAddress
lea     @first,A1
cmpa.l   A1,A0
bne.s   @returnNullEvent

/* set the trap address back to the
* original trap. */
lea     @origTrap,A0
move.l   (A0),A0
move     #GetNextEventTrap,D0
_SetTrapAddress

/* beep to let them know it has been
* removed */
move     #1,-(SP)
_SysBeep

/* free up mem occupied by this patch */
lea     @saveLoc,A0
move.l   A0,A0
_DisposPtr
bra.s    @returnNullEvent

/* This routine will disable _DrawMenuBar
* and _HiliteMenu */
@disableTraps
move     #DrawMenuBarTrap,D0
_GetTrapAddress
lea     @drawMenuBarAddr,A1
move.l   A0,(A1)
lea     @doNothing,A0
move     #DrawMenuBarTrap,D0
_SetTrapAddress
move     #HiliteMenuTrap,D0
_GetTrapAddress
lea     @hiliteMenuAddr,A1
move.l   A0,(A1)
lea     @doNothingWithParam,A0
move     #HiliteMenuTrap,D0
_SetTrapAddress
rts

/* This routine will restore _DrawMenuBar
* and _HiliteMenu and then draw the
* current menu bar */
@restoreMenus
lea     @menus,A0
tst     (A0)
beq.s   @doNothing

```

```

/* unpatch DrawMenuBar and then draw the
 * old menu bar */
    lea    @drawMenuBarAddr,A0
    move.l (A0),A0
    move   #DrawMenuBarTrap,D0
    _SetTrapAddress
    lea    @hiliteMenuAddr,A0
    move.l (A0),A0
    move   #HiliteMenuTrap,D0
    _SetTrapAddress
    lea    @menus,A0
    clr    (A0)
    _DrawMenuBar
@doNothing
    rts

/* This is the _HiliteMenu routine that
 * does nothing. It gets rid of the
 * parameter passed to _HiliteMenu and
 * then returns. */
@doNothingWithParam
    move.l (SP)+,A0
    addq.l #2,SP
    jmp    (A0)

/* This is the routine that actually does
 * the scrolling */
@scrollMenuBar
/* hide the cursor so we don't get part
 * of the cursor scrolled with the
 * menuBar */
    _HideCursor

/* get the current screen width from the
 * Quickdraw global screenBits */
    move.l (A5),A0
/* D3 = # of columns (width) - 1 in the
 * current screen */
    move   screenBits_bounds_right(A0),D3
    sub    screenBits_bounds_left(A0),D3
    move   D3,D2
    subq   #1,D3
/* D2 = # of bytes - 1 in one screen
 * row */
    asr    #3,D2
    subq   #1,D2

/* here's the loop that actually scrolls
 * the menu bar all the way across the
 * screen */
@wayOut
    move   #MENU_BAR_HEIGHT - 2,D1

/* this outside loop will scroll all rows
 * of the menu bar one pixel to the right
 */
@outside
    move   D2,D0
    addq   #1,D0
/* calc # of bytes from base addr to
 * start of current row */
    mulu   D1,D0
    move.l ScrnBase,A0
    adda.l D0,A0
/* white-out the left edge of this row */
    clr.b (A0)
/* D0 = number of words in one row */
    move   D2,D0
    asr    #1,D0
/* set the X flag */
    andi.b #0xEF,CCR
/* this inner loop will scroll one row of

```

```

 * the screen one pixel to the right */
@inside
    roxr   (A0)+
    dbra   D0,@inside

/* go and do the next row */
    dbra   D1,@outside

/* make the corners look like they used
 * to (i.e. rounded and black) */
    move   D2,D0
    move.l ScrnBase,A1
    ori.b  #0xF8,(A1)
    adda   D0,A1
    ori.b  #0x1F,(A1)+
    ori.b  #0xE0,(A1)
    adda   D0,A1
    ori.b  #0x07,(A1)+
    ori.b  #0xC0,(A1)
    adda   D0,A1
    ori.b  #0x03,(A1)+
    ori.b  #0x80,(A1)
    adda   D0,A1
    ori.b  #0x01,(A1)+
    ori.b  #0x80,(A1)
    adda   D0,A1
    ori.b  #0x01,(A1)

    dbra   D3,@wayOut

    _ShowCursor
    rts

/* this routine will print a string in
 * the menu bar, wait a bit and then
 * erase it */
@drawAMessage
/* set the current port to the window
 * manager's */
    move.l (A5),A0
    lea    @savePort,A1
    move.l (A0),(A1)
    move.l WMgrPort,(A0)

/* save the old clipRgn */
    lea    @saveClip,A0
    move.l WMgrPort,A1
    move.l
        OFFSET(GrafPort,clipRgn)(A1),(A0)
/* set the clipRgn to be big enough for
 * the string we want to print */
    subq   #4,SP
    _NewRgn
    move.l WMgrPort,A1
    move.l
        (SP)+,OFFSET(GrafPort,clipRgn)(A1)
    pea    @stringRect
    _ClipRect

/* the string we print depends on the
 * value of the clicks variable */
    lea    @clicks,A0
    move   (A0),D0
    cmpi   #4,D0
    bne.s  @1
    pea    @string1
    bra.s  @5
@1 cmpi   #3,D0
    bne.s  @2
    pea    @string2
    bra.s  @5
@2 cmpi   #2,D0
    bne.s  @3
    pea    @string3

```



```

bra.s      @5
@3      cmpi      #1,D0
        bne.s     @delay
        pea      @string4
@5      move      #10,-(SP)
        move      #14,-(SP)
        _MoveTo
        _DrawString
@delay   lea        @downTime,A0
        move.l     Ticks,(A0)

/* wait until the mouse is released or 60
 * ticks, which ever is longer */
@waitTilMouseUp
    subq.l     #2,SP
    _StillDown
    tst        (SP)+
    bne.s      @waitTilMouseUp
    lea        @downTime,A0
    move.l     (A0),D0
    addi.l     #MESSAGE_DELAY,D0
    cmp.l      Ticks,D0
    bgt.s      @waitTilMouseUp

/* erase the string */
    pea        @stringRect
    _EraseRect

/* dispose of the clipRgn we created */
    move.l     WMgrPort,A1
    move.l
    OFFSET(GrafPort,clipRgn)(A1),-(SP)
    _DisposeRgn
/* restore the old clip rgn */
    lea        @saveClip,A0
    move.l     WMgrPort,A1
    move.l
    (A0),OFFSET(GrafPort,clipRgn)(A1)

/* restore the old port */
    move.l     (A5),A0
    lea        @savePort,A1
    move.l     (A1),(A0)

```

```

rts

@eventRecPtr dc.l      0
/* @menus is TRUE while menus that exist
 * are not being shown */
@menus       dc        0
@downTime    dc.l      0
/* @clicks is the number of menu clicks
 * until menu return */
@clicks      dc        0
/* @words is the number of 16 bit words
 * in one row of screen */
@words       dc        0
/* @saveLoc is the address of our patch
 * in the system heap */
@saveLoc     dc.l      0
@savePort    dc.l      0
@saveClip    dc.l      0
@drawMenuBarAddr dc.l  0
@hiliteMenuAddr dc.l  0

/* this rect should be big enough to
 * enclose the longest of the strings. */
@stringRect  dc
    0,8,MENU_BAR_HEIGHT-1,300

/* define the pascal strings. Does LSC
 * provide a better way than this? */
@string1     dc.b      13,'N','o',
    '','M','e','n','u','s',' ','H','e',
    'r','e'
@string2     dc.b      14,'S','t','i',
    'l','l',' ','N','o',' ','M','e','n',
    'u','s'
@string3     dc.b      7,'N','o','t',
    ' ','Y','e','t'
@string4     dc.b      25,'W','h','e',
    'r','e',' ','D','i','d',' ','T','h',
    'e',' ','s','e',' ','M','e','n','u','s',
    ' ','G','o','?'
@last
    }
}

```



C Workshop

Segmenting DA's



*Tom Saxton
Bellevue, WA
Volume 5, Number 5*

Segmented DAs: Breaking The 32K Limit

In a variety of contexts on the Macintosh we find ourselves running up against "the 32K limit." One area that has been particularly limiting and persistent is the 32K limit on resources containing code. The reason for this has to do with the way the MC68000 addresses memory; specifically one can address code within $\pm 32K$ (and the minus is generally ignored) of some base address very efficiently, so this mode is used for jumping to subroutines and for building jump tables. In particular, Lightspeed C limits the code-containing resources it builds to 32K. This isn't a big problem with applications, since one can break up the code in an application into multiple segments. This segmentation is actually nice in itself, since it allows programs that are too big to fit entirely in memory to run by swapping code resources in and out as needed.

But Desk Accessories and stand-alone code resources (INITs, FKEYs, etc.) are a different story. The Mac OS has nothing built in to handle segmentation for anything but applications, so we're stuck with a 32K size limit. Fortunately, this can all be fixed up with a little inline assembler. I have been working with these sorts of things on several different projects, and have developed a fairly painless way of segmenting either DAs or code resources. This is all done with stand-alone code resources, which I will call PROC's. "Fairly painless" means:

- 1) One shouldn't have to write and debug a bunch of assembly code every time a PROC or routine is added to the project.
- 2) A PROC should be allowed to have multiple entry points. Further, accessible routines should be able to have different numbers of arguments. (A routine with a variable number of arguments is a bit trickier, but not an unreasonable variation on the code given below.)
- 3) Lightspeed's ability to check arguments types via function prototypes should be left intact.
- 3) Calling a procedure in an external PROC should be completely invisible to the calling C routine.
- 4) It should be possible to have multiple entries into a code resource. That is, it should be possible to jump in and out of a given PROC, even to the point of allowing recursion between two PROCs. In short, there should be no more limitations in calling routines than exist in a normal segmented application.
- 5) There should be a mechanism for loading and unloading PROCs the same way application code-resources are handled, or at least a reasonable approximation thereto.

The biggest drawback of doing your own segmentation is that you have to handle part of the link yourself. That is, when you

make a change to one of the PROCs in a project, you have to make it accessible to the "parent" code either by copying the PROC into the parent's resource file, or by having the parent explicitly open resource file(s) containing the PROC(s). At the moment, I am working on a ShareWare DA that will make this process considerably less painful.

The LSC documentation (2.01) supplement shows how to write "glue" to handle a single-entry PROC. Their code breaks down if you want to either call different routines within the PROC, or if there is a possibility of the PROC being called from two different places in the same call chain; more on this below. Working with all of this, I found out several interesting side effects of some ToolBox calls. For instance, `PrJobDialog()` and `PrStdDialog()` both go to the trouble of redrawing any dialog windows in the window list that need updating. This can really hose things if you have user item-drawing routines in a PROC that isn't set up to handle being called again before it returns from the current call.

The Big Issues

There are three basic problems to solve. The first is how to handle multiple entry points. This is the easiest. The Mac PACK resources do this by tacking on an routine selector to the arguments for a routine within the PACK. Upon entry into the PACK, a bit of glue pulls the selector off of the stack and jumps to the right place. A variation on this theme will serve our needs.

Next, LSC (and other environments such as Aztec C) allow non-application code resources to have globals by addressing off of register A4 instead of A5 (which is reserved for applications and QuickDraw). This means that upon entry into a PROC (or a DA) the value of A4 has to be set to the "correct" value to address the PROC's globals. This is relatively easy; the tricky bit is to restore the old value when you exit so that whoever called you can still find their globals. The obvious solution (which LSC describes) is to store the original value of A4 in a fixed location (actually, they use a pc-relative location, which can run afoul of the instruction cache in a 68020, but that's another story). This fails if the PROC gets re-entered from somewhere having a different value of A4, since we can only store one.

The third problem has to do with handling the first two. The simple schemes for handling the first two problems involve three pieces of glue. The first handles putting the routine selector onto the stack and jumping to the (locked!) PROC. The next chunk is at the head of the PROC, it sets up A4, then uses the routine selector to jump to the desired routine. Finally, when the called routine is done, it can't just jump back to the caller; we must go through a last bit of glue to restore the callers's A4. Since, we would like to avoid tacking on this glue at every exit point of

every externally accessible routine, we want to doctor the return address to trick the called routine into returning to a third bit of glue which handles the clean-up. This means we have to pull the real return address off of the stack and store it somewhere. Storing it in a fixed place leads to the same problems we had with A4 above.

The Solution

To avoid storing the two values which need to be saved and recovered in fixed locations, we will store them on the stack. We can't store them below the arguments to the called routine, since then the routine will think it is getting two extra (long word) arguments, and this is supposed to be invisible to the C code. So instead, the first bit of glue shuffles the routine's arguments down on the stack to make room for the 8 bytes of storage we need. We then replace the old return value with the address to the third bit of glue (which does the clean-up). We then jump to the second piece of glue which lives at the entry point to the PROC. It sets up A4 then pulls the selector off of the stack and jumps to the target routine. When the target routine "exits" the clean-up code retrieves the saved values, restores A4 then jumps back to the original caller.

The details of implementing these ideas mainly have to do with doctoring the stack so that both the original caller and the target routine act exactly as if the usual JSR and RTS instructions were used to call and return from the target routine. (Remember this is all supposed to be invisible to the C code.) Care has to be taken to not attempt to address globals while A4 is holding the "wrong" value. We also cannot molest D0 on the way back since that is where any return value is stored.

There are a pair of routines which load and unload a PROC in the spirit of LoadSegment and UnloadSegment. To count as loaded, the PROC resource must be in memory, locked and marked unpurgeable. Unloaded is unlocked and purgeable (don't actually ReleaseResource, in case we want it again). If these PROC's are to be "owned" by a desk accessory, then we have to calculate the resource's actual ID from the driver's ID number and the sub-ID of the PROC. This is all handled by FLoadProc() and UnloadProc().

The following example is a stupid little DA which calls three routines in an external PROC with differing numbers of arguments and return values. Before doing so, it calls FLoadProc() which loads the PROC and stores a pointer to the PROC in a location passed to the routine. FLoadProc() calls MoveHHI() on the resource handle to reduce the possibility of causing fragmentation problems. When the DA is closed (which is the only thing that can be done to or with it), it calls UnloadProc() to unlock and free up the space used by the PROC resource. While debugging, it would be wise to set the pointer to the PROC to -1 when it is unloaded, so that if someone tries to use the unloaded PROC, an address error is immediately generated. This is really only an issue if PROCs are to be loaded and unloaded while the program is running to conserve memory (not an unlikely occurrence, but remember that under LSC, globals are effectively re-initialized when a PROC is released from memory and read back in from disk).

The second source is for the DA. It contains a bare bones DA with routines needed to load and call three routines in a PROC. There are also defines for the procedure selector numbers. These must begin at zero and occur in the same order as the jump table in the PROC. The glue to call a routine in a PROC consists of a "call" to an assembly language macro that does all of the work. Notice that the macro wants the number of WORDs passed as arguments to the routine, not bytes. Note also that this does not count the return address which also gets passed to the routine; the macro takes that into account.

The first source file is for the PROC resource. It contains the entry glue for the PROC. Every routine that is to be externally callable has to have an entry in the JumpTable; see the note about routine selectors above.

To build the PROC, create a new LSC project, add UselessProc.c, set its project type to "Code Resource," its resource type to 'PROC' and its ID to -16000 (sub ID zero owned by DRVR ID 12). Once created, the resource file is named UselessDAProj.rsrc so that it will be copied automatically into the DA file. In general, the PROC resource would have to be copied into the DA's resource file. Note that you don't have to include MacTraps with this project (it does not call any routines defined there).

After building the PROC, and giving its output file the specified name, create a new project called "UselessDAProj", add the first source file and MacTraps to the project, and build the DA. Now Run the project (or open the DA file with Suitcase), and open the DA. The DA will then put up a window (the first PROC call), write something funny in the window (the second PROC call). It will then wait until you close it, at which time it will get rid of its window (the third call to the PROC) and go away. Note that it does not handle even so much as activate or update events. Like I said, useless.

For the Non-Hackers

If assembly language is not your second language already, you're likely thoroughly lost. This should not prevent you from using the trickery here. Below is all you need to know to use the segmenting code.

In the project that calls an external PROC, create a source module which contains the macro definition and a routine for each external procedure you wish to call. Pattern them after the routines OpenWindow(), DrawWindow() and KillWindow() in the DA source file. Each just calls the horrid assembler macro. Be sure to count the words of arguments correctly, or things will die horribly. If you're not sure how to do this, consult your local guru. Next set up a main() in the PROC project patterned after the main() in UselessProc.c. Make one entry in the "JumpTable" for each routine you want to access. Then build a set of defines for the routine selectors (in the same order as they appear in the jump table).

Potential Pitfalls

The easiest things to mess up are counting the words in the argument list. Remember that Pointers, Handles, Points, pointers to structs and arrays, and longs all count 2 words. Ints, shorts,

Booleans and chars all count 1 word (a char is only half a word, but gets passed as a word on the stack, this can be tricky and is probably compiler-dependent). Counting words in a struct or union can be tricky since bytes can get packed together, use sizeof() when in doubt. Don't use sizeof(char), for the reason above. Next is mismatching the order of the routine selectors, check that one twice, too.

Finally, the assembler macro assumes that the glue routine begins with a "LINK" instruction. This is true for any routine that has arguments or local variables. If you want to call an argument-less routine in a PROC, give its glue routine a dummy local variable to force a LINK instruction (LSC talks about LINKs on page 9-2 of the original manual). Alternatively, one could create another version of the macro with the UNLK statement removed for functions with no arguments, but this would leave two chunks of assembler code to maintain.

Debugging Advice

Although I have been using the code for some time and have written it very carefully, if you hit an inexplicable bug while using it, you're going to have to be convinced that the segmentation glue is not at fault. Besides, there is always a chance that I have made some subtle oversight. So, here is some advice for verifying that it's working correctly, or finding out why it is not.

If you haven't already, read LSC's explanation of MacsBug, and the C calling conventions, print out the source for all of the assembler hacking and dive in. If necessary, turn on "MacsBug Symbols" under the Options and recompile the relevant modules so that you can see routine names. The best place to start watching things is just before the call to the glue routine. Watch the arguments as they are pushed onto the stack, step through the glue (not as messy as it sounds), and into the PROC. Make sure the right entry in the jump table is used. As soon as you hit the LINK statement that begins the target routine, do an "mr" (magic return) in MacsBug. You should then break out in the clean-up glue. Step through it and you're back to the caller. The Heap Dump command "hd" is useful for verifying the integrity of the application heap and that the PROC (and its caller!) are correctly loaded into memory, locked in place and unpurgable, use it liberally. It's a good idea to write down the state of the stack, the values for A4 and the return value as the glue is entered for comparison with things before and after the routine in the PROC is called.

The Upshot

This stuff is quite simple and reliable to use once properly set up. By creating multiple PROCs with different sub-IDs, there is virtually no limit to how much code a DA can address (pardon the pun). Routines can even be called directly between PROCs, making it possible to put libraries in PROCs for everyone to use. The only trick is calculating the PROC resource ID, since only the sub ID is known at compile time. The easiest way around that problem is to call an initialization routine for each PROC needing to make inter-PROC procedure calls which stores the base ID in a global.

Listing Useless.c

```
#include <MacTypes.h>
#include <MemoryMgr.h>
#include <Quickdraw.h>
#include <WindowMgr.h>
#include <EventMgr.h>
#include <OSUtil.h>
#include <ResourceMgr.h>
#include <ToolboxUtil.h>
#include <DeviceMgr.h>

#define NULL (0L)
#define drvOpen 0
#define drvPrime 1
#define drvControl2
#define drvStatus 3
#define drvClose 4
#define subidProc 0
enum {
    irtnOpenWindow = 0,
    irtnDrawWindow,
    irtnKillWindow
};

extern short main(CntrlParam*, DctlPtr, short);
extern short DAOpen(CntrlParam*, DctlPtr);
extern short DAClose(CntrlParam*, DctlPtr);
extern WindowPtr OpenWindow(char*, short, short, short, short);
extern void DrawWindow(WindowPtr, char*);
extern void KillWindow(WindowPtr);
extern Boolean FLoadProc(short, Ptr*);
extern void UnloadProc(Ptr*);

Ptr pProc;
short rsidBase;

#define JumpToProc(pPROC, irtn, cwArgs)
asm {
    unlk a6 /* get rid of LSC's frame */\
    movea.l sp, a0 /* address of 1st argument */\
    lea -8(sp), sp /* make room on the stack */\
    movea.l sp, a1 /* bottom of stack */\
    move.w *(cwArgs)+1, d0 /* (# of words to copy)-1 */\
    /* copy rtn addr and arguments */\
LCopyAnother:
    move.w (a0)+, (a1)+ /* move a word down */\
    dbra d0, @LCopyAnother /* done when -d0 == -1 */\
    move.l a4, (a1)+ /* copy our a4 into storage */\
    move.l (sp)+, (a1) /* copy our rtn addr just behind it */\
    pea @LReturnToCaller /* push return for proc */\
    move.w #irtn, -(sp) /* push routine selector */\
    move.l pPROC, -(sp) /* push address of PROC */\
    beq.s @LHandleError /* jump if passed a NULL pPROC */\
LJumpToProc:
    rts /* jump to proc (sleazy trick) */\
LHandleError:
    dc.w 0xA9FF /* let Macsbug "handle" the error */\
    lea 10(sp), sp /* fix up the stack */\
LReturnToCaller:
    lea (2*(cwArgs))(sp), a0 /* get add of storage space */\
    movea.l (a0)+, a4 /* restore our a4 */\
    movea.l (a0), a0 /* get our return address */\
    lea 8(sp), sp /* fix up the stack */\
    jmp (a0) /* return to sender (address known) */\
}

short main(piopb, pdce, drvRoutine)
```

```

CntrlParam *piopb;
DCtlPtr pdce;
short drvrRoutine;
{
    GrafPtr pgpSav;
    short valRtn;

    if (pdce->dCtlStorage == NULL) {
        if (drvrRoutine == 0) { /* open, but no data */
            SysBeep(3);
            CloseDriver(pdce->dCtlRefNum);
        }
        return(0);
    }

    GetPort ( &pgpSav );

    switch (drvrRoutine) {
    case drvrOpen:
        valRtn = DAOpen(piopb,pdce);
        break;
    case drvrControl:
    case drvrPrime:
    case drvrStatus:
        valRtn = 0;
        break;
    case drvrClose:
        valRtn = DAClose(piopb,pdce);
        break;
    }

    SetPort ( pgpSav );

    return valRtn;
} /* main */

short DAOpen(piopb,pdce)
CntrlParam *piopb; /* pointer to parameter block */
DCtlPtr pdce; /* the device control entry */
{
    rsidBase = 0xC000 | (~pdce->dCtlRefNum<<5);

    if ( !FLoadProc ( 0, &pProc ) ) {
        SysBeep(2);
        CloseDriver(pdce->dCtlRefNum);
        return 0;
    }

    /* Using hard-wired window coordinates? For shame... */
    pdce->dCtlWindow=OpenWindow("\Our Window",100,50,300,150);

    ((WindowPeek)pdce->dCtlWindow)->windowKind=pdce->dCtlRefNum;

    DrawWindow(pdce->dCtlWindow,"\Something funny.");

    return 0;
} /* DAOpen */

short DAClose(piopb,pdce)
CntrlParam *piopb;
DCtlPtr pdce;
{
    if ( pdce->dCtlWindow != NULL )
        KillWindow ( pdce->dCtlWindow );

    if ( pProc != NULL )
        UnloadProc ( pProc );
}

Boolean FLoadProc ( subid, ppProc )
short subid;
Ptr *ppProc;
{

```

```

    Handle hProc;

    hProc = GetResource ( 'PROC', rsidBase + subid );
    if ( hProc == NULL ) {
        *ppProc = NULL;
        return FALSE;
    }

    /* avoid fragmentation and other evils */
    MoveHi ( hProc );
    HLock ( hProc );
    HNoPurge ( hProc );
    *ppProc = *hProc;
    return TRUE;
}

void UnloadProc ( pProc )
Ptr pProc;
{
    Handle hProc;

    if ( pProc == NULL )
        return;

    hProc = RecoverHandle ( pProc );
    if ( hProc == NULL )
        return;
    HUnlock ( hProc );
    HPurge ( hProc );
    return;
}

WindowPtr OpenWindow ( stTitle, xLeft, yTop, xRight, yBottom )
char stTitle[];
short xLeft, yTop, xRight, yBottom;
{
    JumpToProc(pProc,irtnOpenWindow,6);
}

void DrawWindow(pwind,st)
WindowPtr pwind;
char st[];
{
    JumpToProc(pProc,irtnDrawWindow,4);
}

void KillWindow(pwind)
WindowPtr pwind;
{
    JumpToProc(pProc,irtnKillWindow,2);
}

Listing: UselessProc.c
#include <MacTypes.h>
#include <MemoryMgr.h>
#include <Quickdraw.h>
#include <WindowMgr.h>
#include <EventMgr.h>
#include <OSUtil.h>
#include <ResourceMgr.h>
#include <ToolboxUtil.h>
#include <DeviceMgr.h>

#define NULL (0L)

extern void main ( void );
extern WindowPtr OpenWindow ( char*, short, short, short,
short );
extern void DrawWindow ( WindowPtr, char * );
extern void KillWindow ( WindowPtr );

void main()
{
    asm {
        movea.l a0,a4 ; get pointer to our globals

```

```

move.w (a7)+,d0 ; pop off routine selector
add.w d0,d0 ; double the selector
add.w d0,d0 ; double it again
lea @JumpTable,a0 ; get address of jump table
jmp 0(a0,d0.w) ; jump to our entry
JumpTable:
    jmp OpenWindow
    jmp DrawWindow
    jmp KillWindow
}
WindowPtr OpenWindow ( stTitle, left, top, right, bottom )
char stTitle[];
short left, top, right, bottom;
{
    Rectrect;

    SetRect ( &rect, left, top, right, bottom );
    return NewWindow(NULL,&rect,stTitle,TRUE,documentProc,-
1L,TRUE,0L);
}

```

```

void DrawWindow ( pwind, stMessage )
WindowPtr pwind;
char stMessage[];
{
    short h, v;

    SetPort ( pwind );
    v = (pwind->portRect.top + pwind->portRect.bottom)/2;
    h = (pwind->portRect.left + pwind->portRect.right)/2;
    h -= StringWidth(stMessage)/2;
    MoveTo ( h, v );
    DrawString ( stMessage );
}

void KillWindow ( pwind )
WindowPtr pwind;
{
    DisposeWindow ( pwind );
}

```



Programmer's Forum

Change the Boot Disk Icon



David Dunham
Goleta, CA
Volume 5, Number 5

I've been asked several times how my *bootDiskIcon* INIT works. I told the questioners that I'd answer in *MacTutor*.

Also, this is a short article. *MacTutor* needs more of them — I know how to initialize Managers and handle events, and I certainly don't need to see pictures in hexadecimal. (This isn't just a snide remark directed at the editor, it's a plea to those with nifty ideas that don't need an entire application to show them off. Write a one or two page article!)

bootDiskIcon is a hack that came about because I don't own an HD20. Apple insists on portraying their internal hard disk as an HD 20 on my desktop. I got so disgusted with this that I wrote an INIT to show something else, anything but an HD 20. ICN# 256, in fact. *bootDiskIcon* is inferior to programs like *Façade* — it's less flexible, and may not work with all hard disks — but it was better than nothing, and it's easy to understand.

The key to *bootDiskIcon* is the way the Finder handles disk icons. A pedantic digression: how does one find out how Finder handles icons? It's not in *Inside Macintosh*. Therefore, it must be in a Technical Note. But Finder isn't a category in the table of contents. So it's off to the index...but it's not listed under "icon." Luckily, it was one of the "Finder" Tech Notes, number 28. How did I know to keep looking? Because I'd seen it years ago when it came out, and knew it was there somewhere. Always skim every Tech Note — not to memorize them, but to remember that they're there.

When the Finder wants a disk icon, it gives the disk driver a Control call with `csCode = 21`. The driver returns a pointer to an ICN#. My first inclination was, why write code when you can change data? The ICN# must be part of the disk driver. The disk driver is part of *Apple HD SC Setup*. I found it there (it's not a resource), changed it, and reinstalled the driver. This failed utterly — there's apparently a checksum.

If I couldn't change the ICN# on disk, how about changing it in memory? I can get its address in memory by giving the driver a `csCode = 21` call, then simply change the data. This is what *bootDiskIcon* does.

Before I get to what you're all waiting for, the source code, I'll mention the difference between patches and INITs. A patch is a change to a piece of code, almost always one of the traps. An INIT resource is any piece of code that runs at startup time (typically in a file of type INIT, cdev, or rdev). An INIT frequently installs a patch, but it could do anything at all, like playing a tune or subverting a disk driver's icon.

More ideological baggage: INITs should announce their presence. Paul Mercer has a good way to do this, called *ShowINIT*. I'm told the source code's available on AppleLink. I've modified it to handle color icons, and compiled the whole thing as a separate resource, *SHOW 0*. *bootDiskIcon* calls this

code. You're welcome to use it in any non-commercial (free or shareware) programs — just use *ResEdit* to paste *SHOW 0* into your own INIT file. You may not use it in public domain programs. These are those without a copyright holder, and I don't want my code in the public domain. Don't make the mistake of calling your free program "public domain." Once you do, you can't get it back (despite what Adobe thinks).

We can almost get to the code, but you might be wondering how to test it. It's real inconvenient to reboot, just to see if a change worked. Luckily, you don't have to. Remember that an INIT is just a piece of code that happens to get executed at boot time. You can often execute the same code at any time, and instruct your development system to make an INIT resource when you're finished debugging. In *bootDiskIcon*'s case, I simply executed it, juggled into the Finder layer, and verified that I'd changed the disk's icon.

OK, you want the code. It starts out by calling my *SHOW* resource — the parameters are reversed from the way they're defined because my *ShowICON* is defined as a Pascal procedure. My sloppiness in not defining it as such in this C program probably works only because this is an INIT — if I trash the stack, it doesn't much matter, since I'll be returning to INIT 31 very soon.

The real work begins with finding out the driver `refNum` and drive of the first mounted disk (now you know why it's called *bootDiskIcon* — I told you it was a quick hack). I then find the address of the driver's ICN#, get an ICN# of my own choice, and overwrite the driver's using *BlockMove*.

```
#include <deviceMgr.h>
#include <hfs.h>

#define NIL0L

void main(void);          /* Always use prototypes! */

void main() {
    Ptr      icn;
    int      error;        /* Shouldn't really be write-only. */
    HParamBlockRec pb;
    cntrlParam cpb;
    Handle    handle;

    /* Show icon on the bottom of the screen */
    handle = GetResource('SHOW', 0); /* Get handle to the PROC */
    if (handle == 0L) { /* Something's wrong (couldn't load) */
        SysBeep(32);    /* Let somebody know */
        return;         /* Don't try to call it! */
    }
    HLock(handle);        /* Hold down the PROC */
    (**(ProcPtr *)handle)(-1, 128);
    /* ShowICON() - note reversed parameters */
    HUnlock(handle);      /* Let it float in the heap again */
}
```

```

/* Change the icon */
pb.volumeParam.ioNamePtr = NIL;
pb.volumeParam.ioVolIndex = 1; /* First mounted volume */
error = PBHGetVInfo(&pb,FALSE);
cpb.ioRefNum = pb.volumeParam.ioVDRefNum;
cpb.ioVRefNum = pb.volumeParam.ioVDrvInfo;
/* drive (ioDrvNum) */
cpb.csCode = 21;
error = PBControl(&cpb,FALSE);
icn = *(Ptr *)&cpb.csParam[0];
handle = GetResource('ICN#',256);
BlockMove(*handle,icn,256L);
}

```

Known Bugs

Actually, these are rumored bugs, because it works for me! I'm told it doesn't work with Jasmine disks where the driver puts the SCSI ID inside the flower. It probably doesn't work with a boot floppy, because the Finder normally special cases the disk icon. (Furthermore, Finder probably doesn't consider floppies to be "foreign drives.")



HindCight

Lisa's Hourglass Is Back!



Mike Morton
Waipahu, HI
Volume 5, Number 5

Back to the Future:

The Lisa's hourglass cursor comes to life

In the rivalry between Lisa and Macintosh at Apple, many Lisa features weren't carried over into Mac development: a large screen, multi-tasking, stationery pads, virtual hardware, a huge keyboard... Now that these features are slowly reappearing in Macs, it's time to bring back one more dead Lisa feature — the hourglass cursor — and also breathe a little life into it. (Let's just not bring back rectangular pixels.)

This article actually shows you the low-level mechanics of three kinds of animated cursors:

- a one-handed watch, a simple sequence of ordinary cursors
- a two-handed watch, built by combining cursors
- an hourglass with falling sand: "drawn" by the application

There's been a lot of discussion (see [Further Reading](#)) of whether an animated cursor should be done by a VBL (timed) task so the main program doesn't have to keep calling it. When I originally wrote these routines some years back (in MDS assembler), I made them a VBL task. I'm no longer sure that's a good idea. So this package doesn't animate on its own, but it should be easy to write a timed task to call them.

Some notes on VBL cursor animation

In case you do want to make the cursor run on its own, here are some things to keep in mind. (Thanks to Seth Lipkin for telling me about a number of these surprises...)

- Remember that when a VBL task wakes up, the globals pointer (A5) isn't guaranteed and memory management can't be performed. Further, under MultiFinder, the CurrentA5 variable may hold someone else's A5! You should store your A5, if you need it, adjacent to your task's VBL entry.

- There's a low RAM location called CsrBusy. As I understand it, when this byte is non-zero, you shouldn't do a SetCursor — I suspect it means there's a SetCursor already in progress. If your VBL task sees this flag, it should probably just try again in one tick.

- The low-memory CsrVis flag is cleared when the cursor's not visible (it's been hidden or temporarily obscured). Again, call back later.

- Probably the biggest problem with VBL tasks is that they keep running no matter what. If your application locks up, the user will still see some life, and may not realize what's going on. If you must use the VBL trick, you might consider making the same task function as a "watchdog timer", trying to figure out if the application is doing anything. If the application fails to bump some counter at least every *n* seconds, the task could hint at this with a different cursor. It's very risky to tell the user any other way, since any dialog would mean memory management. (I'm

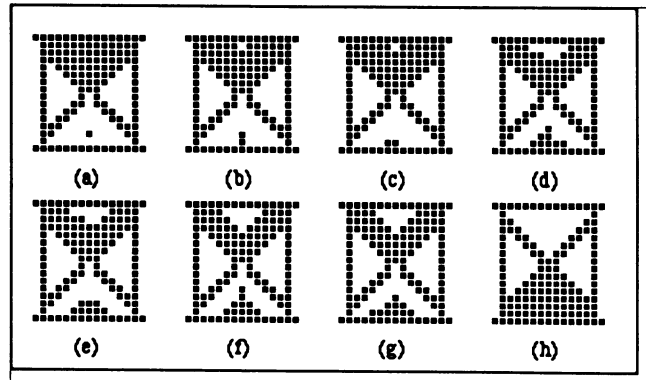


Figure 1. Details of building the hourglass

not sure if a VBL task can use the Notification Manager...)

Drawing methods

So if this package isn't going to run the cursor on its own, what good is it? Well, it does various kinds of drawing, from simple to tricky. Each of the three cursor types has a different drawing method.

The watch cursor functions a lot like the Finder's. There's a single list of cursors which are shown in a cycle. This could be done with a Finder-style ACUR resource, but if you want to do

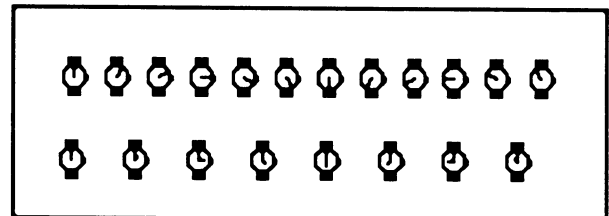


Figure 2. Components of watch cursor

the animation with a timed task, you don't want to do GetResource calls. (Actually, you could load the resources and make them non-purgeable before starting. But another reason is that all the data used by this watch is also used by the two-handed variant.)

There are 12 frames to this animation sequence, shown in the top part of Figure 2.

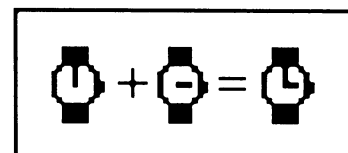


Figure 3. Assembling a two-handed watch

The two-handed watch is only slightly more complicated. It uses two sets of data, one for the minute hand and one (in the bottom of Figure 2) for the hour hand. The hour hand has only 8 positions; I couldn't draw 12 hands that small and keep it looking nice. So don't worry if it looks like "four-thirty o'clock".

Figure 3 shows how the hour and minute hand are logically "or"-ed together to make a two-handed watch. Like a real clock, the hour hand's position advances only once per revolution of the minute hand.

The hourglass is a lot more complicated. Looking at parts of Figure 1 shows some examples of how a grain of sand, a black pixel, moves: At any given time, a single grain of sand is moving (a). When it lands on top of another grain (b), it randomly falls to one side or the other (c). If it lands with only one side open (d), it falls to the open side (e). If it lands with several steps to fall (f), it falls all the way until it doesn't have an opening on either side below (g). [Note that the bottom line of the hourglass must already be solid black pixels, or the sand will try to "leak out".] Finally, when a grain of sand drops as far as it can, and it's above a certain level (h), the bottom chamber is full, and the cycle starts over.

Every movement by the grain takes a single animation frame. A similar process is "stealing" a grain from the top chamber. This follows a similar path upwards, but walks the whole path at once each time it needs a new grain — see the comments below on the stealGrain function.

The sample driver program

This main program isn't a shining example of Macintosh programming style. It's just meant to show all the functionality of the subroutines.

Menu commands let you select from among the three types of cursors; control the speed; and quit. That's about it.

The structure of the program is simple. The main loop calls the cursor animation routine at idle time, also calls SystemTask, and then does simple event handling — just enough to operate the menus.

One thing to note while using the program is the flicker. On a Macintosh Plus or SE, the screen refresh is linked to the tick count. Even though the task is not implemented as a VBL task, the cursor update takes place near the beginning of a tick. This is apparent on the Plus or SE because the cursor flickers more when it's near the top of the screen. On a Mac II, where the monitor's refresh rate isn't sync'd with the ticker, the cursor flickers sporadically at any point on the screen.

A feature I included in the original (ca. 1986) program was a way to measure how much CPU time the cursor animation takes up. The program would see how many times it could execute a loop with the cursor running at various speeds, and not running at all. I haven't done the measurements for this LightspeedC version, but that assembler version took up less than 2% of the CPU time if the animation was 15 times per second or less. Animating at 60 times per second could take up to around 7% of the time. You have to watch out for this, since your application is presumably showing the cursor to apologize for some slow operation in the first place!

[A side note: the time used is dependent on the cursor's horizontal position, since SetCursor has to shift the image into position — the function is a sawtooth curve with a 16-pixel period on a Mac Plus or SE. This took a long time to find...]

Finally, note that "main.c" includes "AnimCurs.h", which contains the prototypes specifying the interface to the package.

The subroutines

Here's a quick tour through the contents of the subroutine package.

Data structures: Each of the three cursor variants has a struct defined for it, remembering the state of the drawing so it can be updated. Then there's a type "cursInfo", which is a union of all three, plus the current cursor, timing information, and the variable controlling which type of cursor is being shown.

Note that this structure has to remember the whole state of the cursor. For the watches, it just has to remember one or two hand positions. For the hourglass, it must keep track of whether or not there's a falling grain of sand, and the current column (stored as a bit) and row the sand is at. Unlike with the other types, the cursor data itself is part of the information, because some routines examine its bits to see where the sand should fall.

Data: I've included all the masks and data in-line. They probably ought to be resources. If they were, some care would be needed if the routines were being called from a VBL task.

Functions:

The three initialization (acStartxxx) routines all do about the same thing: set up the delay, the type, the cursor mask and data, and type-specific information.

Two utility routines (copy16, or16) are used to copy and combine the "Bits16" type, which holds a cursor's mask or data.

hideGrain and showGrain just change the visibility of the current grain of sand. They're a good example of how the state information is self-contained; they need no parameters. moveGrain just updates the position by hiding, moving, and showing the grain.

coinFlip is kind of cool. I didn't want to use QuickDraw's random number generator. (In addition to needing A5, it can mess up an application's random sequence by stealing elements from it nondeterministically.) So I used a method which is simplified from the engine used in the "DissBits" subroutine. Although the random numbers used in that subroutine are poor, the random bits used by coinFlip() are quite random. [See Knuth in Further Reading for details.]

stealGrain takes a random pixel from the top chamber and turns it white. It works by percolating through black pixels until it reaches the top or runs out of black pixels. It's a lot like the way a single black pixel percolates down through white ones, but it percolates all at once when a new grain is needed. dropGrain just steals a grain from the top, then sets up a new grain and marks it as moving.

Three routines advance specific cursor types. nextWatch and nextWatch2 are very simple. nextHour drops a grain if needed and then advances the current grain by one pixel. doN_extFrame just cases out on the current cursor type to call one of these three.

Two final routines are called from the application: `acNext` should be called as often as possible. When it's time to do so, it computes and displays the next cursor. `acDelay` can be called to set the delay, in ticks, between frames.

Projects for a rainy day

I've kept the scope of this project small. Here are some varied ways you could take it from here:

- Intercept `GetResource` calls for type ACUR and CURS in someone else's application. When they think they're just displaying a sequence of cursors, you could be computing and displaying something much more complex, such as the hourglass.

- Sort of the opposite: Intercept `SetCursor` calls and record the cursors and the delays in between them. Automatically build and save an ACUR resource. Then anything drawn "on the fly" (like the hourglass) can be transcribed as an ACUR and stored for simple programs (like the Finder) to use. Of course, these may turn out quite large.

- Experiment with changing the mask in the hourglass so the glass part is "transparent". You'll have to modify the mask as the data portion is modified, too.

- Add an arrow type to the set of cursors supported, so `acNext()` can always be called at idle, whether or not there's animation going on. Hint: if the cursor variant is an arrow, `acNext()` does nothing.

- Handle color cursors.

- Time the routines and choose an optimal animation rate to keep the animation smooth yet lessen flicker and not use too much CPU time. Try it on various CPUs and monitors to check both performance and appearance.

- Write a VBL task which makes the cursor animate automatically. Have it notice when another part of the program does a `SetCursor` trap, and assume this means it should turn itself off. (In an older program, I examined the low-memory cursor data to see if the cursor had changed; probably a bad move for color cursors...)

- Put everything into a driver, so any development system can use it. Could you have it get time during `SystemTask` calls, so the application needn't call it?

Further reading

For more on the method used in the `coinFlip()` routine, see:

- Donald Knuth, The Art of Computer Programming, 2nd ed., Vol II: Seminumerical Algorithms, pp. 29-31

- Mike Morton, "A Digital Dissolve Effect for Bitmapped Graphics Screens", Dr. Dobbs's Journal, November 1986

- Mike Morton, "The DissBits Subroutine", MacTutor, December 1985

- William H. Press, et al, Numerical Recipes: The Art of Scientific Computation, pp. 209-213

Knuth's writeup is the most interesting as far as a theoretical basis. The Recipes includes a useful table of constants. My DDJ article is more in-depth as far as applying the method.

For more on the vertical retrace manager, see:

- "The Vertical Retrace Manager", Inside Macintosh, Vol. II, pp. 347-354

- Robert B. Denny, "Using the Vertical Retrace Manager", August 1985

For more on animation (of cursors or anything), see:

- Dick Chandler, "VBL Task Animation", MacTutor, February 1989

- Stuart Gordon, dir., "Re-Animator", 1985

- Letters, MacTutor, March 1988, p. 12

- Mike Morton, "Alternate Video Screen Animation", MacTutor, June 1986

- David Stoops, "Animated Cursors", Letters, MacTutor, December 1988, pp. 108-109

Conclusions

We've covered three ways to animate cursors — two simple ones based on resource-like data and one which does the drawing more "by hand". Doing the drawing directly into a cursor with bit manipulation is much faster than using `QuickDraw` to set and clear pixels. It's acceptable because the format of a monochrome cursor isn't about to change.

The animation routines are decoupled from the issue of whether the animation should be invoked from an idle routine or from a timed task. Timed tasks bring the risk that a user gets no indication of when the application is hung up, along with various technical subtleties. If you do want to make a task animate your cursor, it might be a good idea to make the task also check for signs of life in the application.

Listing: AnimCurs.h

```
/*Prototypes for the animated cursor subroutines.
   See "AnimCurs.c" for details on how to call these.
*/
```

```
void acStartHG (int delay); /* set up for hourglass cursor */
void acStartW1 (int delay); /* set up for one-handed watch */
void acStartW2 (int delay); /* set up for two-handed watch */
```

```
void acDelay (int delay); /* set delay between frames */
```

```
void acNext (void); /* idle-time call: advance cursor */
```

Listing: AnimCurs.c

```
/*Animated Cursor subroutines.
```

Copyright © 1989 by Michael S. Morton, all rights reserved.
Written February '89 for MacTutor. Feel free to use these or modify them so long as the original author and source are noted.

Routines included are:

- o `acStartHG` - set the cursor to be an hourglass
- o `acStartW1`, `acStartW2` - set the cursor to be a 1- or 2-handed watch
- o `acDelay` - set the delay, in ticks, between animation steps
- o `acNext` - idle-time routine

To set up any of these cursors, call the `acStart` routine, passing the delay you'd like between frames. Then call `acNext` during your idle routine. To change the delay, call `acDelay`. To get a different kind of cursor, stop calling `acNext` and use `SetCursor` to get it.

```
*/
/*State information for three cursor types: */
typedef struct /* info on the HOURGLASS: */
( long coin; /* used in random number generator */
  int grainRow; /* row number (0..15) of falling grain */
  int grainBit; /* bit mask for falling grain */
  Boolean grainMoving; /* flag: is the grain still moving? */
) hourInfo; /* define this type */
```

```
typedef struct /* info on ONE-HANDED WATCH: */
( int wFrame; /* which frame of animation are we on? */
) watch1Info; /* define this type */
```

```
typedef struct /* info on TWO-HANDED WATCH: */
( int mFrame; /* frame of minute hand */
  int hFrame; /* frame of hour hand */
) watch2Info; /* define this type */
```

```
/*Generalized cursor type: */
typedef enum
( hgType, w1Type, w2Type ) cursType; /* selects one of
above structs */
```

```
typedef struct /* tracks any kind of cursor */
( Cursor curs; /* current cursor */
  int delay; /* delay between frames (ticks) */
  long nextFrame; /* time of next frame (ticks) */
  cursType type; /* type of cursor */
  union /* depending on type of cursor: */
  { hourInfo hg; /* o info on hourglass */
    watch1Info w1; /* o info on one-handed watch */
    watch2Info w2; /* o info on two-handed watch */
  } u; /* give union an easy name */
) cursInfo, *ciPtr;
```

```
/*cursor information is used globally all through file. */
cursInfo theInfo; /* sole instance of this struct */
```

```
/*16x16 data and masks for cursors. Some of these are OR'd
together to make the two-handed watch. Others are just
backgrounds which are drawn on, such as for the hourglass. */
int hourData [16] = /* hourglass, about to start falling */
( 0xFFFE, 0x7FFC, 0x7FFC, 0x7FFC, 0x5FF4, 0x4FE4, 0x47C4,
  0x4384, 0x4284, 0x46C4, 0x4C64, 0x5834, 0x701C, 0x600C,
  0x4004, 0xFFFE );
int hourMask [16] = ( 0xFFFE, 0x7FFC, 0x7FFC, 0x7FFC, 0x5FF4,
  0x4FE4, 0x47C4, 0x4384, 0x4384, 0x47C4, 0x4FE4, 0x5FF4,
  0x7FFC, 0x7FFC, 0x7FFC, 0xFFFE );
```

```
int watchMask [16] = ( 0x3E00, 0x3E00, 0x3E00, 0x3E00, 0x7F00,
  0xFF80, 0xFF80, 0xFFC0, 0xFFC0, 0xFF80, 0x7F00, 0x3E00,
  0x3E00, 0x3E00, 0x3E00, 0x0000 );
```

```
int watchMinutes [12][16] = /* twelve minute-hand positions */
(
  ( 0x3E00, 0x3E00, 0x3E00, 0x3E00, 0x4900, 0x8880, 0x8880,
    0x88C0, 0x80C0, 0x8080, 0x4100, 0x3E00, 0x3E00, 0x3E00,
    0x3E00, 0x0000 ),
  ( 0x3E00, 0x3E00, 0x3E00, 0x3E00, 0x4300, 0x8480, 0x8480,
    0x88C0, 0x80C0, 0x8080, 0x4100, 0x3E00, 0x3E00, 0x3E00,
    0x3E00, 0x0000 ),
  ( 0x3E00, 0x3E00, 0x3E00, 0x3E00, 0x4100, 0x8180, 0x8680,
    0x88C0, 0x80C0, 0x8080, 0x4100, 0x3E00, 0x3E00, 0x3E00,
    0x3E00, 0x0000 ),
  ( 0x3E00, 0x3E00, 0x3E00, 0x3E00, 0x4100, 0x8080, 0x8080,
    0x8FC0, 0x80C0, 0x8080, 0x4100, 0x3E00, 0x3E00, 0x3E00,
    0x3E00, 0x0000 ),
  ( 0x3E00, 0x3E00, 0x3E00, 0x3E00, 0x4100, 0x8080, 0x8080,
```

```
0x88C0, 0x86C0, 0x8180, 0x4100, 0x3E00, 0x3E00, 0x3E00,
  0x3E00, 0x0000 ),
  ( 0x3E00, 0x3E00, 0x3E00, 0x3E00, 0x4100, 0x8080, 0x8080,
    0x88C0, 0x84C0, 0x8480, 0x4300, 0x3E00, 0x3E00, 0x3E00,
    0x3E00, 0x0000 ),
  ( 0x3E00, 0x3E00, 0x3E00, 0x3E00, 0x4100, 0x8080, 0x8080,
    0x88C0, 0x88C0, 0x8880, 0x4900, 0x3E00, 0x3E00, 0x3E00,
    0x3E00, 0x0000 ),
  ( 0x3E00, 0x3E00, 0x3E00, 0x3E00, 0x4100, 0x8080, 0x8080,
    0x88C0, 0x90C0, 0x9080, 0x6100, 0x3E00, 0x3E00, 0x3E00,
    0x3E00, 0x0000 ),
  ( 0x3E00, 0x3E00, 0x3E00, 0x3E00, 0x4100, 0x8080, 0x8080,
    0x88C0, 0x80C0, 0xC080, 0x4100, 0x3E00, 0x3E00, 0x3E00,
    0x3E00, 0x0000 ),
  ( 0x3E00, 0x3E00, 0x3E00, 0x3E00, 0x4100, 0x8080, 0x8080,
    0x8FC0, 0x80C0, 0x8080, 0x4100, 0x3E00, 0x3E00, 0x3E00,
    0x3E00, 0x0000 ),
  ( 0x3E00, 0x3E00, 0x3E00, 0x3E00, 0x4100, 0xC080, 0x8080,
    0x88C0, 0x80C0, 0x8080, 0x4100, 0x3E00, 0x3E00, 0x3E00,
    0x3E00, 0x0000 ),
  ( 0x3E00, 0x3E00, 0x3E00, 0x3E00, 0x6100, 0x9080, 0x9080,
    0x88C0, 0x80C0, 0x8080, 0x4100, 0x3E00, 0x3E00, 0x3E00,
    0x3E00, 0x0000 );
```

```
int watchHours [8][16] = /* eight hour-hand positions */
(
  ( 0x3E00, 0x3E00, 0x3E00, 0x3E00, 0x4100, 0x8880, 0x8880,
    0x8880, 0x8080, 0x8080, 0x4100, 0x3E00, 0x3E00, 0x3E00,
    0x3E00, 0x0000 ),
  ( 0x3E00, 0x3E00, 0x3E00, 0x3E00, 0x4100, 0x8080, 0x8480,
    0x8880, 0x8080, 0x8080, 0x4100, 0x3E00, 0x3E00, 0x3E00,
    0x3E00, 0x0000 ),
  ( 0x3E00, 0x3E00, 0x3E00, 0x3E00, 0x4100, 0x8080, 0x8080,
    0x8E80, 0x8080, 0x8080, 0x4100, 0x3E00, 0x3E00, 0x3E00,
    0x3E00, 0x0000 ),
  ( 0x3E00, 0x3E00, 0x3E00, 0x3E00, 0x4100, 0x8080, 0x8080,
    0x8880, 0x8480, 0x8080, 0x4100, 0x3E00, 0x3E00, 0x3E00,
    0x3E00, 0x0000 ),
  ( 0x3E00, 0x3E00, 0x3E00, 0x3E00, 0x4100, 0x8080, 0x8080,
    0x8880, 0x8880, 0x8880, 0x4100, 0x3E00, 0x3E00, 0x3E00,
    0x3E00, 0x0000 ),
  ( 0x3E00, 0x3E00, 0x3E00, 0x3E00, 0x4100, 0x8080, 0x8080,
    0x8880, 0x9080, 0x8080, 0x4100, 0x3E00, 0x3E00, 0x3E00,
    0x3E00, 0x0000 ),
  ( 0x3E00, 0x3E00, 0x3E00, 0x3E00, 0x4100, 0x8080, 0x8080,
    0x8880, 0x8080, 0x8080, 0x4100, 0x3E00, 0x3E00, 0x3E00,
    0x3E00, 0x0000 ),
  ( 0x3E00, 0x3E00, 0x3E00, 0x3E00, 0x4100, 0x8080, 0x9080,
    0x8880, 0x8080, 0x8080, 0x4100, 0x3E00, 0x3E00, 0x3E00,
    0x3E00, 0x0000 );
);
```

```
/*Function prototypes for stuff we define externally: */
#include "AnimCurs.h"
```

```
/*Function prototypes for our static stuff: */
void acStartW (cursType type, int delay); /* start either kind
of watch */
void copy16 (int *src, Bits16 *dst); /* copy 16x16 bit image */
void or16 (int *src, Bits16 *dst); /* OR a 16x16 bit image */
void hideGrain (void); /* hide current grain in hourglass */
void showGrain (void); /* show current grain in hourglass */
Boolean coinFlip (void); /* produce a pseudo-random boolean */
void stealGrain (void); /* remove a grain from top section */
void dropGrain (void); /* get a grain moving */
void moveGrain (int dh, int dv); /* move a grain in transit */
void nextHour (void); /* compute next cursor: hourglass */
void nextWatch (void); /* watch */
void nextWatch2 (void); /* 2-handed watch */
void doNextFrame (void); /* compute next frame and show it */
```

```
/*acStartHG () - Initialize the hourglass cursor, with the
specified delay between animation frames. This is an external
```

```

routine. */
void acStartHG (delay)
    int delay;          /* INPUT: pause between frames */
{ theInfo.delay = delay; /* record delay in structure */
  theInfo.nextFrame = TickCount () + delay; /* figure when to
animate next */
  theInfo.type = hgType; /* remember the type: hourglass */
  theInfo.u.hg.grainRow = -1; /* make a grain drop in
dropGrain () */

  /*Initialize cursor data, mask and hotspot (altho dropGrain
does data, too) */
  copy16 (hourData, & (theInfo.curs.data));
  copy16 (hourMask, & (theInfo.curs.mask));
  theInfo.curs.hotSpot.v = 8; /* stick hotspot... */
  theInfo.curs.hotSpot.h = 8; /* ...in middle */

  dropGrain ();          /* start first grain dropping */
} /* end of acStartHG () */

/*acStartW1 () - Just like acStartHG, but with a one-handed
watch. This is an external routine.
*/
void acStartW1 (delay)
    int delay;          /* INPUT: pause between frames */
{ acStartW (w1Type, delay);
} /* end of acStartW1 () */

/*acStartW2 () - Just like acStartHG, but with a two-handed
watch. This is an external routine.
*/
void acStartW2 (delay)
    int delay;          /* INPUT: pause between frames */
{ acStartW (w2Type, delay);
} /* end of acStartW2 () */

/*acStartW () - Start a one- or two-handed watch. */
static void acStartW (type, delay)
    cursType type;      /* INPUT: w1Type or w2Type */
    int delay;          /* INPUT: pause between frames */
{ theInfo.delay = delay; /* record delay in structure */
  theInfo.nextFrame = TickCount () + delay; /* figure when to
animate next */
  theInfo.type = type; /* remember the type */

  if (type == w1Type) /* init different types: */
      theInfo.u.w1.wFrame = 0; /* one-handed: set to zero */
  else
      ( theInfo.u.w2.mFrame = 0; /* two-handed: set minute */
        theInfo.u.w2.hFrame = 0; ) /* ...and hour frame to zero */

  /*Initialize the cursor data, mask and hotspot: */
  copy16 (watchMinutes [0], & (theInfo.curs.data));
  copy16 (watchMask, & (theInfo.curs.mask));
  theInfo.curs.hotSpot.v = 8; /* stick hotspot... */
  theInfo.curs.hotSpot.h = 8; /* ...in middle */
} /* end of acStartW () */

/*copy16 () - Copy an array of 16 ints into a Bits16 structure
(the data or mask of a cursor. */
void copy16 (src, bits)
    register int *src;   /* INPUT: source data */
    Bits16 *bits;       /* OUTPUT: where to stuff it */
{ register int *dst = (int *) bits; /* coerce to a handier
data type */
    int i;              /* FORTRAN-ish loop counter */

    for (i = 0; i <= 15; i++) *dst++ = *src++; /* copy 16 words */
} /* end of copy16 () */

/*or16 () - Just like copy16(), but we OR, not COPY. */
void or16 (src, bits)
    register int *src;   /* INPUT: source data */

```

```

    Bits16 *bits;       /* OUTPUT: where to stuff it */
{ register int *dst = (int *) bits; /* coerce to a handier
data type */
    int i;              /* FORTRAN-ish loop counter */

    for (i = 0; i <= 15; i++) *dst++ |= *src++; /* OR 16words */
} /* end of or16 () */

/*hideGrain () - Hide pixel where the current grain is. */
static void hideGrain ()
{ theInfo.curs.data [theInfo.u.hg.grainRow] &=
  (~ theInfo.u.hg.grainBit);
}

/*showGrain () - Show the pixel where current grain is. */
static void showGrain ()
{ theInfo.curs.data [theInfo.u.hg.grainRow] |=
  theInfo.u.hg.grainBit;
}

/*coinFlip () - Return a pseudo-random boolean value. We use
shift-register system detailed in Knuth, among other places.
*/
static Boolean coinFlip ()
{ register Boolean result;

  if (! theInfo.u.hg.coin) /* zero? */
      theInfo.u.hg.coin = 1; /* avoid demon state */
  result = (theInfo.u.hg.coin < 0); /* note if high bit set */

  theInfo.u.hg.coin <<= 1; /* shift old value over 1 bit... */
  if (result) /* ...and if top bit was '1'... */
      theInfo.u.hg.coin ^= 0xc5; /* ...then futz with low bits */

  return (result);
} /* end of coinFlip () */

/*stealGrain - Grab a grain of sand from the top chamber of
the hourglass. We want to percolate a white pixel up from the
starting bit position. */
static void stealGrain ()
{ register int v = 6; /* start at row 6, tip of top half */
  register int hBit = 0x0100; /* start at the middle */
  register Boolean done = false; /* haven't finished percolat-
ing yet */
  register int aboveWord; /* row of sand grains above us */
  register Boolean leftOK, rightOK; /* are there bits to
steal above? */

  while (! done)
      (-v; /* assume we can move up a row */
        aboveWord = theInfo.curs.data [v]; /* get that row from
the cursor image */

        /*Decide which direction to steal from - if there's a
grain right above us, we'll take that. Otherwise, we have to
choose between the ones diagonally up from us. */
        if ((aboveWord & hBit) == 0) /* have a bit above us? */
            ( leftOK = (0 != (aboveWord & (hBit << 1))); /* have
someone above on left? */
              rightOK = (0 != (aboveWord & (hBit >> 1))); /* someone
above on right? */

              /*Branch, depending on which way(s) we can go: */
              if (leftOK && rightOK) /* both available? */
                  ( if (coinFlip ()) hBit <<= 1; /* yes: flip a coin... */
                    else hBit >>= 1; /* ...to decide */
                  )
              else if (leftOK) hBit <<= 1; /* just left: take it */
              else if (rightOK) hBit >>= 1; /* just right: take it */
              else ( ++v; done = true; ) /* neither: back down */
            ) /* end of no bit above us */

            if (v <= 1) done = true; /* if we hit top row, we have to

```

```

stop */
}
/* end of while-not-done */

theInfo.curs.data [v] &= (~ hBit); /* snuff this bit */
}
/* end of stealGrain () */

/*dropGrain () - Get next grain of sand to fall. The "moving"
flag is false. We get one grain of sand from the top chamber.
If the bottom chamber is filled, we reinit the cursor to a
full top chamber. Either way, we flag that a grain is moving,
and draw it.
*/
static void dropGrain ()
{ stealGrain (); /* pull a grain from the top chamber */

/*If the pile in the bottom chamber has filled it, the most
recent grain dropped didn't get very far. */
if (theInfo.u.hg.grainRow < 9) /* are we piled too high? */
    copy16 (hourData, & (theInfo.curs.data)); /* yes:
redraw all */

theInfo.u.hg.grainRow = 8; /* set row number of new grain */
theInfo.u.hg.grainBit = 0x0100; /* set bit in row */
theInfo.u.hg.grainMoving = true; /* we're cooking with gas
now */
showGrain (); /* show the grain */
}
/* end of dropGrain () */

/*moveGrain () - Move the current grain by a given delta.
Note that a positive "dh" is a right shift. */
static void moveGrain (dh, dv)
int dh, dv; /* INPUT: amount to move by */
{ hideGrain (); /* hide where it is now */

theInfo.u.hg.grainRow += dv; /* update vertical position */
if (dh > 0) /* update horizontal position */
    theInfo.u.hg.grainBit >>= dh;
else theInfo.u.hg.grainBit <<= -dh;

showGrain (); /* show where it is now */
}
/* end of moveGrain () */

/*nextHour () - Advance the hourglass by one frame. */
static void nextHour ()
{ register int v, hBit; /* coordinate and bit pos of grain */
register int nextWord; /* word of grains below us */
register Boolean leftOK, rightOK; /* flags for OK
directions to fall */

if (! theInfo.u.hg.grainMoving) /* grain hit bottom
last time? */
    dropGrain (); /* yes: get a new one */

v = theInfo.u.hg.grainRow; /* grab position and bit just... */
hBit = theInfo.u.hg.grainBit; /* ...for easier typing
& efficiency */

nextWord = theInfo.curs.data [v+1]; /* get word below this
one */
if ((nextWord & hBit) == 0) /* slot below us free? */
{ moveGrain (0, 1); /* yes: just move down one */
return; /* and that's all we need */
}

leftOK = (0 == (nextWord & (hBit << 1))); /* have nobody
below on left? */
rightOK = (0 == (nextWord & (hBit >> 1))); /* nobody below
on right? */

/*Branch, depending on which way(s) we can go: */
if (leftOK && rightOK) /* can fall left or right? */
{ if (coinFlip ()) /* yes: randomly choose: */
    moveGrain (1, 1); /* o fall to the right */
else moveGrain (-1, 1); /* o fall to the left */
}
}

else if (leftOK) moveGrain (-1, 1); /* left only: do it */
else if (rightOK) moveGrain (1, 1); /* right only: do it */
else theInfo.u.hg.grainMoving = false; /* can't move: stop;
reset next time */
}
/* end of nextHour () */

/*nextWatch - Compute the next frame of one-handed watch
animation. */
static void nextWatch ()
{ ++ theInfo.u.w1.wFrame; /* bump frame ahead by 1 */
if ((theInfo.u.w1.wFrame) >= 12) /* outside of range 0...11? */
    theInfo.u.w1.wFrame = 0; /* yes: wrap around */

copy16 (watchMinutes [theInfo.u.w1.wFrame],
& (theInfo.curs.data));
}
/* end of nextWatch () */

/*nextWatch2 - Compute the next frame of two-handed watch
animation. */
static void nextWatch2 ()
{ ++ theInfo.u.w2.mFrame; /* bump MINUTE frame ahead by 1 */
if ((theInfo.u.w2.mFrame) >= 12) /* outside of range 0...11? */
    theInfo.u.w2.mFrame = 0; /* yes: wrap around */

copy16 (watchMinutes [theInfo.u.w2.mFrame],
& (theInfo.curs.data));

if (theInfo.u.w2.mFrame == 0) /* MINUTE frame just hit 12
o'clock? */
{ ++ theInfo.u.w2.hFrame; /* yes: bump HOUR frame ahead
by 1 */
if ((theInfo.u.w2.hFrame) >= 8) /* outside of range 0...7? */
    theInfo.u.w2.hFrame = 0; /* yes: wrap around */
}

/*Add hour-hand image into minute-hand. */
or16 (watchHours [theInfo.u.w2.hFrame],
& (theInfo.curs.data));
}
/* end of nextWatch2 () */

/*doNextFrame - Compute next frame, dispatching on type of
cursor. */
static void doNextFrame ()
{ switch (theInfo.type)
{ case hgType: nextHour (); break;
case w1Type: nextWatch (); break;
case w2Type: nextWatch2 (); break;
default: break;
}
}
/* end of switch on cursor type */
}
/* end of doNextFrame () */

/*acNext-Idle-time animation. This is an external routine.*/
void acNext ()
{ if (TickCount () < theInfo.nextFrame) /* time to animate
yet? */
    return; /* nope: return */

theInfo.nextFrame = TickCount () + theInfo.delay; /* reset
wake-up call */
doNextFrame (); /* draw next frame internally */
SetCursor (& (theInfo.curs)); /* and display it */
}
/* end of acNext () */

/*acDelay - Set delay, in ticks, between frames. This is an
external routine.
*/
void acDelay (delay)
int delay; /* INPUT: new delay */
{ theInfo.delay = delay;
theInfo.nextFrame = TickCount () + delay; /* figure when to
animate next */
}
/* end of acDelay () */

```

Listing: main.c

/*Animated Cursor demonstration main program.

Copyright © 1989 by Michael S. Morton, all rights reserved.
Written February '89 for MacTutor.

*/

#include "AnimCurs.h" /* define functions in package */

/*Internal prototypes: */

void main (void);

void makeMenus (void);

void doMenu (long result);

int theDelay = 4; /* delay between animation frames, tix */

enum /* items in our sole menu */

(mnWatch = 1, mnWatch2, mnHourG, mnUnused1,
mnFaster, mnSlower, mnTop, mnUnused2, mnQuit);

void main ()

(EventRecord evtRec; /* info from GetNextEvent () */
int inWhat; /* from FindWindow (): where was click? */
WindowPtr theWin; /* unused - just for FindWindow () */
long menuResult; /* from MenuKey () or MenuSelect () */

/*Usual initializations: perhaps more than we need here. */
InitGraf (&thePort); /* initialize QuickDraw */
InitFonts (); /* init the Font Manager */
FlushEvents (everyEvent, 0); /* ignore pending events */
InitWindows (); /* init the Window Manager */
InitMenus (); /* init the Menu Manager */
TEInit (); /* init TextEdit */
InitDialogs (0L); /* init Dialog Mgr; no restart proc */
InitCursor (); /* show standard arrow cursor */

makeMenus (); /* set up one cheap little menu */
acStartW1 (theDelay); /* initialize to a one-handed watch */

/*Main loop: at idle time, update the cursor. Check for
menu commands via the mouse or keyboard. */

while (1)
{ acNext (); /* idle time: animate the cursor */
SystemTask (); /* can't hurt, altho we don't do DAs */

if (GetNextEvent (everyEvent, & evtRec)) /* try for an
event */
{ if (evtRec.what == mouseDown) /* click? */
{ inWhat = FindWindow (evtRec.where, &theWin); /* look
up where it hit */
if (inWhat == inMenuBar) /* menu click? */
doMenu (MenuSelect (& (evtRec.where))); /* track &

process menu hit */
else SysBeep (2); /* can't handle clicks elsewhere */
} /* end of handling mouseDown */

else if (evtRec.what == keyDown) /* keypress? */
doMenu (MenuKey ((char) evtRec.message)); /* don't
require cmdKey */
} /* end of handling one event */

} /* end of infinite loop */
} /* end of main () */

/*Make menu in-line. (Resources? This is just a demo
driver...) */

void makeMenus ()

{ MenuHandle theMenu; /* our sole menu */

theMenu = NewMenu (1, "\pMenu");
AppendMenu (theMenu, "\pWatch/W;Two-handed watch/2;Hour-
glass/H;(-");
AppendMenu (theMenu, "\pFaster/F;Slower/S;Top speed/T;(-");
AppendMenu (theMenu, "\pQuit/Q;(-");
AppendMenu (theMenu, "\pAnimated Cursor Demo;written in
LightspeedC");
AppendMenu (theMenu, "\pCopyright © 1989;Michael S.
Morton;for MacTutor");
InsertMenu (theMenu, 0); /* put it in the menu bar */
DrawMenuBar (); /* show our handiwork */
} /* end of makeMenus () */

void doMenu (result)

long result; /* INPUT: result from MenuSelect/Key */
{ int command = LoWord (result); /* extract item number */

switch (command)
{ case mnWatch: acStartW1 (theDelay); break;
case mnWatch2: acStartW2 (theDelay); break;
case mnHourG: acStartHG (theDelay); break;

/*Note that we can't let the delay go down to zero for
"Faster". */

case mnFaster: if (theDelay > 1) -theDelay; acDelay
(theDelay); break;
case mnSlower: ++theDelay; acDelay
(theDelay); break;
case mnTop: theDelay = 1; acDelay
(theDelay); break;

case mnQuit: ExitToShell (); break;
default: break; /* ignore unrecognized items */
} /* end of switch on command */

HiliteMenu (0);

} /* end of doMenu () */



abC

Window Structures



*Bob Gordon
Minneapolis, MN
Volume 5, Number 6*

Welcome to the second incarnation of the MacTutor intro column. The column is called abC because of its introductory nature (the "abc's"), and it uses the C language (the "C"). What I propose to do is to continue in the spirit of the previous set of articles, but focus less on C and more on the Mac.

I also will not attempt to present a complete program each time. Since an introduction by its very nature tends to be a bit repetitive, I found the last time that the same code would appear for months at a time. This took up a lot of space in the magazine, and made it difficult to focus on one issue as we were always carrying around a large amount of other code just to make an application work. This means that the articles will tend to be shorter, which will likely make it more possible for me to get them done.

One other point before we start. I found that one of the most useful things in writing for MacTutor is getting mail. Feel free to write; comments are much appreciated; and include your phone number.

Making a Window

One of the first steps in a Mac application is putting a window on the screen, typically in response to a request from the user. This method developed here is done entirely in code (as opposed to using resources). This has some limits, but offers the ability to easily add some features to ease the maintenance of windows.

There are two points: First a window consists of the window stuff and the application specific stuff. From the Mac's point of view, the window stuff consists of only the frame and does not even include the scroll bars (if any). On the other hand the application probably does not want to be concerned with such things as the scroll bars, zoom boxes, sizing, et cetera.

Second, it is good programming practice to separate the application specific code from the more generic user-interface code. This eases maintenance and debugging and modification. To this end I created a file, shell.c which handles many of the generic properties, helper files (e.g. windowhelp.c) to add the capabilities I wanted, and the file app.c, which actually contains the parts of the application. In the snippets of code that follow, you will notice an occasional call to a function like appxxx(). This is a call in the shell part of the program to the app.c file. It is here, after, for example, preparation has been made for updating a window (appupdate()), that the application must update the application portion.

windowhelp.h

The windowhelp functions work because the WindowRecord defined in the ToolBox is a fixed length object. What I did

is add some additional data structures on the end. This simply makes the object a little longer, but all the toolbox window and QuickDraw functions will work correctly. The advantage of doing this is that I can now keep additional information, which I think should be attached to a window in a safe place where I can always find it, and the application parts of the code need to concern themselves with it.

One might suggest that the WRefCon (a field in the WindowRecord specifically for the application's own use) field is the proper place for this. This is probably true, but so much of this stuff seemed to either be more part of the window than the application (scroll bars) or likely to be part of any application (knowing whether the contents had changed or not), that it seemed a Good Idea to put it all together in one place. In this way much of the Mac user interface is handled for us each time we start a project. I use the WRefCon field to store the reference to the applications information proper.

Now, I make have no pretenses that this is the optimal way of doing things. In fact, I know there are additions and changes I am going to make, but if we wait to get all that worked out, we'll never get this written.

A few comments about the fields:

The first four are what I call window margins. I noticed that many applications place various graphic objects around the edges of the window. There are scroll bars to the bottom and right, rulers on the top and tools on the left. Whatever it is, it brings in the edge of the window as far where the application can write. The window margins are sizes, in pixels of the areas reserved by things around the edges. The application can change them (if the user asks to see the rulers, for example).

The next field, cursrt, is the rectangle defined by the window and the margins. I called it cursrt because outside the rectangle the cursor is typically the 11 o'clock arrow, and inside the cursor is whatever the application would like.

The zoomrt is the rectangle to hold the size to zoom to if the user clicks the zoom box. Frankly, I haven't implemented zooming yet. It may be that we'll want two rectangles for zooming: one to hold the maximum size and one to hold the small (user set) size.

mouser is a code for the mouse cursor. I made it a char simply because I couldn't imagine anyone wanting more than 255 cursors and because the built-in cursors are numbered one through four. If you want to add a cursor resource, simply start the numbers at five and everything will work fine. And if you want more cursors or want to use bigger numbers, make it a short.

The changed field simply states whether or not the window contents have changed.

The next field, ckind (contents kind) I used to indicate the

kind of window. Currently this is either text or a object drawing. Whether this will stay this way or not I don't know. Currently I use this in the functions that handle update and activation events to determine what to do.

The last field is a handle to the current TextEdit record. A window may have more than one. This is needed because when a window is activated, the blinking cursor needs to be placed in whichever edit record the user left it in.

As you can see, much of this information falls somewhere between what is strictly application specific and what is part of the window. Much of it, however, the application either need not be concerned with or is needed for almost all applications.

Following the WindowStruct definition are several groups of #defines. The most interesting of these are the WDOC/WGROW/WZOOM. By using these when you define a window (simply add together the ones you want), you can specify whether you want a simple document window to have a grow box or a zoom box. You can also add in the codes for the vertical or horizontal scroll bars.

```

/*windowhelp.h */
#include "TextEdit.h"

#ifdef WINDOWHELP_H
#define WINDOWHELP_H

/* window add-on structure */

#define WindowStruct struct w_struct
WindowStruct
{
    WindowRecord wr;      /* the original window record */
    uchar    mtop;        /* margin indents */
    uchar    mleft;
    uchar    mbottom;
    uchar    mright;
    Rect     cursrt;      /* rectangle used for cursor control */
    Rect     zoomrt;      /* rectangle for zoom operation */
    char     mouser;      /* mouse pointer id (cursor shape) */
    char     changed;     /* if window contents havebeen changed */
    short    ckind;       /* kind of contents */
    TEHandle curtext;     /* handle current te rec for window */
};

#define Woffset    18
#define SBarWidth  15

/* these definitions allow easy generation of the four square
 * cornered titled windows. The basic (simplest) window is the
 * WDOC (NoGrowDocProc). To this optionally add WGROW to add a
 * grow box and/or WZOOM to add a zoom box.
 */
#define WDOC    4
#define WGROW   -4
#define WZOOM    8
#define WVBAR   16
#define WHBAR   32

/* specify the content of the window */
#define CDRAW    1
#define CTEXT    2

/* Value in windowKind field of WindowRecord by WindowNew
 * if the window has a grow box. This is used by routines that
 * redraw the window to decide whether to draw the grow box
 * or not

```

```

*/
#define HASGROW  9

#endif

```

WindowHelp.c

WindowHelp.c consists mainly of two functions, WindowNew() and WindowClose().

WindowNew() opens a window after creating a WindowStruct as shown above (no check for memory yet). Note that it only gets four arguments. The title (which can be a null string in which case the window is named "Untitled"), a percentage (the window is opened as a certain percentage of the monitor size), the windowkind (WDOC/WGROW/WZOOM), and the content kind.

A few other notes: The call to AppPage() is a kludge. It is supposed to return the page size. Currently some numbers are just hard coded in AppPage(). The page size is needed for scrolling. Just put in sizes for an 8.5 by 11 piece of paper (in pixels) or whatever size page you want. The call to AppNew() is where the application does the initialization of its own data for a new window and places a reference to it (typically a handle) in the WRefCon.

WindowClose() closes a window. After checking to see if the window is a desk accessory (and handling that), it calls AppClose() to allow the application to close its own structures. Then it closes the window and frees the memory. It is in here that we would check to see if the window contents had changed (and had not been saved) and ask the user if we were to save the file.

```

/* windowhelp.c
 * provides some routines to aid in opening and closing
 * windows
 */

#include "abc.h"
#include "Quickdraw.h"
#include "EventMgr.h"
#include "WindowMgr.h"
#include "MenuMgr.h"
#include "windowhelp.h"
#include "controlhelp.h"
#include "cursorhelp.h"

short w_total = 0; /* counts total number of windows opened */
short w_count = 0; /* counts number of windows actually open */

WindowNew(title,percentsz,windkind,contkind)
char *title;
short percentsz;
short windkind;
short contkind;
{
    char *thetitle;
    Rect boundsrt;
    WindowRecord *w;
    WindowStruct *ws;
    short width;
    short depth;
    short leftedge;
    short topedge;
    Rect pagesz;
    short hasscroll;

```

```

hasscroll = windkind & WVBAR + WHBAR;
/* mask out scroll bar options */
windkind &= 8;
if (title)
    thetitle = title;
else
    thetitle = "Untitled";
SetRect(&boundsrt, screenBits.bounds.left + 4,
        screenBits.bounds.top + 24,
        screenBits.bounds.right - 4,
        screenBits.bounds.bottom - 4);
width = boundsrt.right - boundsrt.left;
depth = boundsrt.bottom - boundsrt.top;
AdjustRect(&boundsrt,
    topedge = 30 + Woffset * w_total,
    leftedge = 30 + Woffset * w_total,
    ((depth * percentsz) / 100) - topedge,
    ((width * percentsz) / 100) - leftedge);
ws = (WindowStruct *)NewPtr(sizeof (WindowStruct));
w = (WindowRecord
*)NewWindow(ws, &boundsrt, CtoPstr(thetitle), True,
    windkind, (WindowPtr)-1, True, 0);
ws = (WindowStruct *)w;
switch (windkind)
{
    case documentProc :
    case zoomDocProc :
        w->windowKind = HASGROW;
        break;
};
if (w->windowKind == HASGROW) /* set window margins */
{
    ws->mright = ws->mbottom = SBarWidth;
    AppPage(&pagesz);
    if (hasscroll == WHBAR)
        ScrollNew(w, 0, 0, pagesz.right, HORZ);
    if (hasscroll == WVBAR)
        ScrollNew(w, 0, 0, pagesz.bottom, VERT);
    DrawControls(w);
    DrawGrowIcon(w);
}
else
    ws->mright = ws->mbottom = 0;
ws->mtop = ws->mleft = 0;
ws->changed = FALSE;
ws->ckind = contkind;
ws->curtext = NIL;
SetCursorRect(w); /* set cursor rect for cursor control */
AppNew(w); /* function to open application structure */
w_count++;
w_total++;
PtoCstr(thetitle);
}

/* WindowClose()
* Closes the window passed to it. Assumes window was opened
* with WindowNew(). If window is a desk accessory, it will
* be closed correctly. If it is an application window, prior
* to the window close there is a call to AppClose() (which
* the application must supply) to allow any application
* specific closing.
*/
WindowClose(w)
WindowRecord *w;
{
    WindowStruct *ws;
    short refnum;

    if (ws = (WindowStruct *)w)
        if ((refnum = w->windowKind) < 0)
            CloseDeskAcc(refnum);
        else
        {
            AppClose(w); /* close app storage, must do before */

```

```

CloseWindow(w); /* CloseWindow */
DisposPtr(w);
w_count--; /* decrement the window count */
if (w_count == 0)
    /* if the number of windows on screen goes to */
    w_total = 0;
    /* zero, set total to zero so next time we */
    /* open a window it will start at the original */
} /* window position */

/* AdjustRect
* adjusts coordinates of a rectangle by four separate
* amounts. adjustment is in (towards center) if amounts are
* positive and out if amounts are negative;
*/
AdjustRect(rec, t, l, b, r)
Rect *rec;
short t, l, b, r; /* values to adjust rectangle by */
{
    rec->top += t;
    rec->left += l;
    rec->bottom -= b;
    rec->right -= r;
}

/* RtGlobalToLocal
* Adjusts a rectangle to local coordinates
*/
RtGlobalToLocal(rt)
Rect *rt;
{
    Point *pt;

    pt = (Point *)rt;
    GlobalToLocal(pt);
    pt++;
    GlobalToLocal(pt);
}

```

The last little piece of code illustrates how these functions are called from the shell program. `dofile()` is part of the code that responds to menus.

```

dofile(item)
short item;
{
    WindowRecord *w;

    switch(item)
    {
        case iNew : WindowNew("", 40, WDOC+WGROW+WVBAR, CTEXT);
            break;
        case iClose : WindowClose(FrontWindow());
            break;
        case iQuit : while (w = (WindowRecord *)FrontWindow())
            {
                WindowClose(w);
            }
            ExitToShell();
            break;
    }
}

```

I hope this gives you some ideas on how to handle windows. Since it can open as many windows as you want, there should be some check for memory limits. I also hope this is suitably introductory. Again, if you have any questions or comments, please write.



C Workshop

Sub and Superscripting with TE



Robert McKenzie
Santa Ana, CA
Volume 5, Number 6

SUPERSCRIPTING AND SUBSCRIPTING IN TE

Robert McKenzie formerly of Mountain Lake Software, Inc. as the project manager of network, file, and multitasking programming tools for ClassKit™, a C++ object-oriented application framework. His career experience of 15 years ranges from programming parallel processor mainframes to desktop computers. For the past two and one half years he has been programming on the Macintosh with emphasis at the network and driver level.

Introduction

This article describes how to encrypt a super/subscripting scheme into the data structures of the new Text Edit manager routines. Adding this scheme requires that one or two trap hooks be intercepted. This is also shown.

It was good to see Apple take a quantum leap from the original TE manager to the new TE manager. But with all the capabilities now built into TE, including color support and style runs, it's hard to understand why they left out super/subscripting.

An application which finally provoked me to look for a way to add this functionality to TE was a test generation program for teachers called MAKETEST®. In particular, teachers who need to indicate power/base notation need super/subscripting, unless of course they were to be limited to some Fortran-like notation, such as:

$$x^3 + y = 5$$

Data Structures in TE

The method I will describe depends on the new TE manager routines. These routines provide the run styles which are needed to keep the necessary style information on a character by character basis. Trying to manage character by character style information in the old TE manager routines would require a completely different approach.

```
STElement:
  stCount:  INTEGER;
  stHeight:  INTEGER;
  stAscent:  INTEGER;
  stFont:    INTEGER;
  stFace:    Style;
  stSize:    INTEGER;
  stColor:   RGBColor;
```

The data structure of interest is called an STElement (V-262). A style element in TE is the collection of individual styles common to a sequence of characters in the TE text buffer. Any sequence of characters that share the same combination of styles,

whether the sequences are adjacent or not, share the same style element entry in the TE manager's data structures.

As the TE manager updates the screen it checks where one run ends and another begins. Each time the styles of a character sequence change, the grafport's font and text information are set to the combination of fonts and styles found in the style element entry for the new style run.

The TE manager does not draw characters one at a time. Once it has set the grafport's text parameters to the information contained in the style run, it calls the QuickDraw routine 'DrawText' to draw all the characters within the style run at one time.

Now we have a couple of pieces of information gathered which begin to suggest an approach for supporting super/subscripting within TE:

1) STElements is where style combinations are held. This is where we need to find a place to store the super/subscripting flags.

2) TE always uses 'DrawText' for displaying one or many characters. This can be verified by using the trap intercept functionality of TMON. By trying to intercept various traps while forcing TE to output to the screen in all the ways it can, you can determine that only DrawText is used.

3) The grafport must be set with all the correct font and style information if the characters are to be drawn correctly with DrawText.

If we are going to use the style element structure to store the super/subscripting flags, we must find some unused bits within one of the fields: stCount, stHeight, stAscent, stFont, stFace, stSize, or stColor. Super/subscripting requires a 3 state representation: super/sub/normal. Therefore, we need 2 bits which can represent up to 4 states.

The stStyle field seems like a likely place to look, especially since it is the field used to represent the standard styles. However, being a 7-bit set type it only leaves 1 unused bit. Not enough.

The stCount, stHeight, stAscent, and stSize fields contain numeric values used to compute character spacing for any given combination of styles. If we tried to encode super/subscripting in one of these fields it would alter the values computed in the character spacing calculations. The unpredictability of when one of these fields might be used makes these fields particularly messy to use.

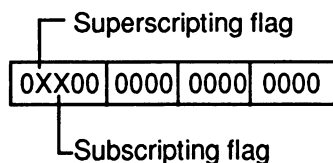
It might be possible to use the stColor field if we imposed some constraints on the color schemes that could be used. For example, if we used bits 0 and 1 within the stColor field, that would mean that the other entries in the color table would have to be organized such that if 0-2 were added to it we would still get the same color. Otherwise, for example, if a user was typing with

green characters and chose *only* super/subscripting from the menu, the super/subscripted character might come out as, say, red instead of green. Of course you'd have to be very familiar with the Color manager in order to create a color table to avoid this problem. (See the Color manager of Inside Mac-V for more information on the Color manager.)

This leaves only the stFont field as a place for these super/subscripting bits to live. To know if this is possible we have to know how the font number is used. The most important point is how the font number is used to look up the character characteristics of the various fonts. When QuickDraw is called to output text, as when DrawText is called, it passes the font number to the Font manager to get the font description information. The font number is checked against fonts that are already loaded. If the font is loaded, a handle to the font description information is given back to QuickDraw for its use. If it isn't loaded, then the font number is given to the Resource manager and, again, a handle is returned to QuickDraw. Great. With this knowledge we now have a well defined event in which an attempt to use the field we have altered can be intercepted, the super/subscripting bits temporarily masked, and the normal TE character processing sequence can then go on. Of course using 2 bits within the stFont field also causes constraints on the range of usable font values. Without tabulating all the existing font numbers in the world this constraint seems quite reasonable.

Choosing the Super/Subscripting Bits

The font number is a 16-bit integer which allows for approximately 32K fonts. That's a lot of fonts for people to choose from. Most of the fonts that I know of are assigned increasing values starting at 0 (0 is the systemfont), so obviously we don't want to use any of the low ordered bits within the font field. It's also the case that many programmers assign a field to be a negative value when its meaning is to be temporarily or permanently changed. Therefore, I chose not to use bit 15 either. The two best bits to use are bits 13 and 14 of the integer field.



The corresponding value for each bit is:

13: 8192

14: 16384

This means that no fonts with numbers between 8192 and 16384 can be used if we reserve these bits for super/subscripting. Does anybody know of such fonts? (My knowledge of font number assignment is limited.) Of course, giving the user a choice of over 8K fonts to choose from is still plenty.

Setting the Super/Subscripting Bits

Setting the super/subscripting bits is done quite easily, and in a straightforward manner using the normal TE manager style setting function. An example is shown in the 'doEditing' func-

tion in the source listing given.

First I defined some C constants which define the 3 states of super/subscripting:

```

SUPER  0x4000 /* THE 14th BIT */
SUB    0x2000 /* THE 13th BIT */
REGULAR 0x1FFF /* BITS 0 - 12 */

```

When the user chooses one of these states, the existing font information is retrieved by using the function:

```

TEGetStyle(**TEH).setStart-1,
           &Styl, &lght, &ascent, TEH)

```

In each case the old super/subscripting information is cleared with the bitwise AND operator:

```
Styl.tsFont &= REGULAR;
```

In the case of normal scripting this is all that needs to be done before resetting the style. If super or subscripting is chosen, the appropriate bit is set with an OR operator:

```

Styl.tsFont |= SUPER;
or
Styl.tsFont |= SUB;

```

Then all that needs to be done is to set the new style:

```
TESetStyle(doFont, &Styl, 1, TEH);
```

Shifting characters up or down

To make use of the bits that we just set we need to hook into the QuickDraw DrawText function. It is when this function is called that we will make our move and fool the TE manager into shifting the characters appropriately. Hooking into the DrawText trap will be explained in the next section.

But once our intercept routine has been called what do we do? We make use of point 3 from the 'Data Structures in TE' section which states that the grafport is set correctly just before the text is about to be drawn. When set, the grafport's font field will contain the font value set in the Styl.tsFont field above including our super/subscripting bits. Therefore, the hook routine can get the font value from the grafport, test it for having either the super/subscripting bit set, and adjust the grafport's pen location appropriately. (See Figure 2) The complete routine is then:

```

DrawTextHook(...)
...
{
    register int fnt, offset;
    GrafPtr     gptr;

    GetPort(&gptr);
    fnt = gptr->txFont;

    offset = ((fnt & SUPER)
              ? -3
              : ((fnt & SUB)
                 ? 3

```

```

        : 0));

if (offset)
{
    Move(0,offset);
    TextFont(fnt & REGULAR);
}

DrawText(text,pos,cnt);

if (offset)
{
    Move(0,-offset);
    TextFont(fnt);
}
}

```

Notice the first TextFont call resets the grafport's font to the expected font by stripping the subscripting bits. The text is drawn and, if needed, the pen loc and grafport font is restored to the state previously set by the TE manager.

Intercepting the DrawText/GetResource traps

Setting up trap hooks requires knowledge of assembly and stack management, particularly when mixing languages like I have done in this project. My programming was done under MPW using assembly and the GreenHill C compiler. Trap intercepting is itself a topic of interest. Here I will explain the important points as related to super/subscripting.

The DrawText hook is used to call our DrawTextHook function described above. The GetResource hook is used to keep QuickDraw from being confused by a strange font value found in a style run. The GetResource hook will be called as the TE manager tries to setup the grafport with the character attributes of a style run before it calls DrawText. Trying to set up the grafport with an unknown font number is the same well-defined event as described earlier.

When GetDrawTrapInfo and GetResTrapInfo are called at application startup time they merely calculate trap information, such as the amount of data space the trap needs, its code size, and its beginning location. These values are used to set up the hook functions in the system heap as required.

As seen in the assembly source listing, the data space definitions only force the assembler to output the correct code. Actual allocation of data space for the hook routine must be done by the SetupHook function. The actual trap code is between the TrapBeg and TrapEnd labels. This is the code copied down into the system heap.

Neither of these trap hooks is reentrant. Therefore, a call-level variable, 'CLvl', must be maintained in order to keep track of how deeply any calls to these routines are nested. If the CLvl becomes greater than one, then it should do nothing except act like the standard trap.

When the toolbox calls the DrawText trap, we have to take care of transforming the Pascal-oriented parameters on the stack to MPW-C parameters. The Pascal call to DrawText has the form:

```
DrawText(textPtr,firstByte,byteCount);
```

where:

textPtr: is a 32-bit pointer
firstByte: is a 16-bit integer
byteCount: is a 16-bit integer

And, of course, the parameters get pushed onto the stack from left to right which would leave the stack in the condition:

Return Address: RRRR 4 bytes
byteCount: BB 2 bytes
firstByte: FF 2 bytes
TextPtr: PPPP 4 bytes

low mem ... : RRRR : BBFF : PPPP : ... high mem
|
A7

When calling a C routine, integers must be treated as 32-bit integers. Therefore the DrawText hook has a little stack management to do. (See Figure 1) First it saves the return address of the original caller and adjusts the stack pointer. Then it temporarily copies the byteCount value into D0 without altering the stack pointer. Then to transform the firstByte value to a 32-bit integer, it simply zeros out the high order word of the stack. Now the stack contains only the firstByte and textPtr parameters. Next the byteCount value in D0 is extended to be a 32-bit integer and its value is pushed back on the stack.

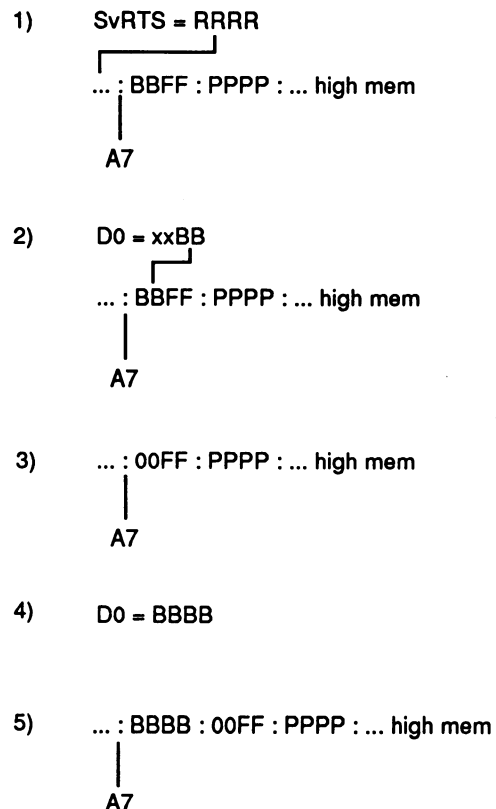


Figure. 1

C parameters are pushed on the stack from right to left. Instead of trying to correct the ordering problem on the stack, and unnecessarily wasting time, we can just take advantage of this knowledge and declare our DrawTextHook with its parameters in the reverse order. Therefore the omitted parameters to DrawTextHook above would be:

```
DrawTextHook(cnt,pos,text)
    int    cnt;
    int    pos;
    char *text;
    ( ...
    )
```

I was writing and finishing this article just as the latest tech notes from Apple were issued. Tech Note #207 describes additional functionality added to the TE manager in System 6.0. The significant addition with respect to this article is the TEDrawHook. It provides exactly the same kind of functionality that was required when writing the hook routine to intercept DrawText. The TEDrawHook function is provided with the same parameters, plus some, that is needed by the DrawTextHook function. Therefore, the DrawTextHook could be used as the definition for this now supported TE hook.

The essence of the GetResource hook is much simpler. The only thing we have to do here is to keep the Font manager from returning bogus font information when given an altered font number with our super/subscripting bits set.

We know that the Font manager is trying to look up font description information when it calls GetResource with a resource type of 'FOND'. Therefore, any time GetResource is called with this type, and following our constraint that no fonts in the range 8192 to 16384 will be allowed, then all the GetResource hook has to do is mask out the 13th and 14th bit of the font number where it sits on the stack. (See Figure 2) Once these bits are cleared, then we just let the Resource manager continue to do its thing.

Summary Remarks

A scheme to add the functionality of super/subscripting was presented. However, there are many alterations that can be made to this scheme in order to refine the functionality into a more word processing-like functionality. For example super/subscripted characters in this basic scheme are printed in the same point size and with a fixed offset of 3 points. A more sophisticated approach would be to reduce the point size of shifted characters and allow the user to choose how many points to offset the shifted characters.

However, beware that if these extra capabilities are added, you will also have to handle having the correct font description resources loaded correctly and, perhaps, reposition the pen loc more carefully. Otherwise, character blurring may occur.

Listing: DoMake

```
make > MakeMiniEdit
MakeMiniEdit
```

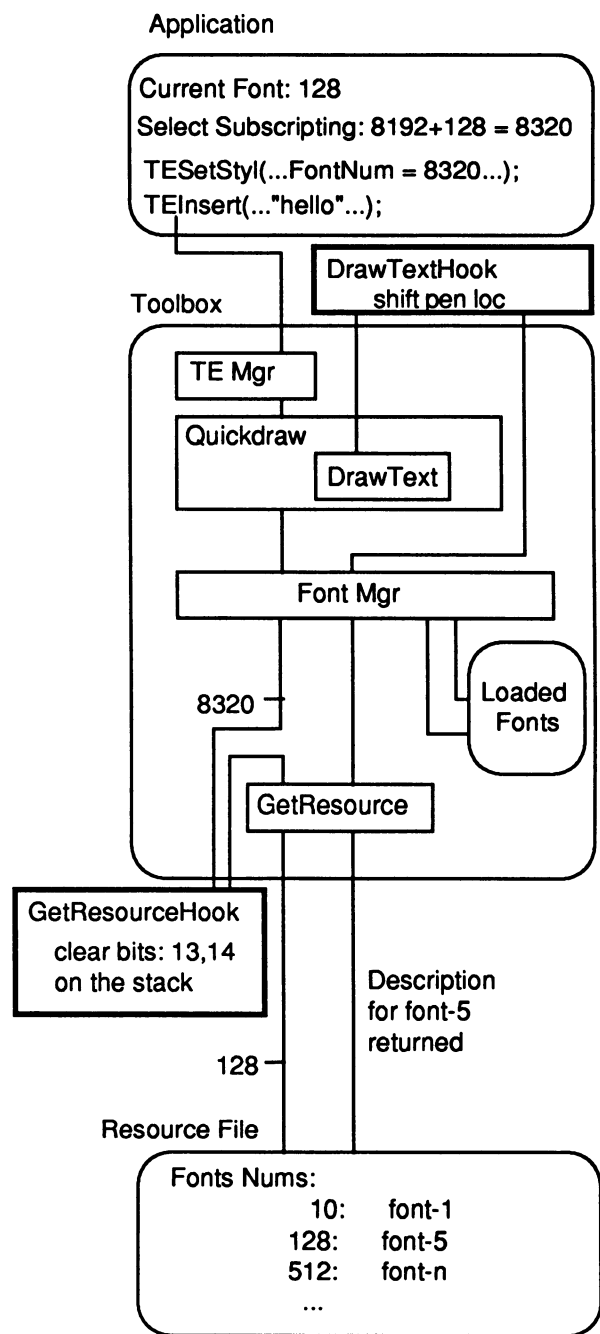


Figure 2

Listing: MakeFile

```
MiniEdit ff windows.c.o
MiniEdit.c.o      d
SuperSub.c.o      d
TrapDefs.a.o      d
TrapHook.c.o
link -bf d
windows.c.o      d
MiniEdit.c.o      d
SuperSub.c.o      d
TrapHook.c.o      d
TrapDefs.a.o      d
"(libraries)runtime.o" d
```

```

"(libraries)cinterface.o" @
"(libraries)interface.o" @
-o MiniEdit

windows.c of windows.c @
MiniEdit.h
c -g windows.c

MiniEdit.c.o f MiniEdit.c @
MiniEdit.h @
TrapHook.h
c -g MiniEdit.c

SuperSub.c.o f SuperSub.c @
TrapHook.h
c -g SuperSub.c

TrapHook.c.o f TrapHook.c @
TrapHook.h
c -g TrapHook.c

TrapDefs.a.o f TrapDefs.a
asm -case obj TrapDefs.a

```

Listing: TrapDefs.a

```

;*****
;
; TrapDefs.a
;
;*****

PRINT      OFF
INCLUDE    'Traps.a'
INCLUDE    'SysEqu.a'
PRINT      ON,OBJ

CODEREFS FORCEJ2T

StackFrame RECORD (A6Link),DECR
  DataSizePtr: ds.1 1
  TrapSizePtr: ds.1 1
  TrapAddrPtr: ds.1 1
  retaddr:     ds.1 1
  A6Link:      ds.1 1
  stacksize:   EQU *
ENDR
; =====

GetDrawTrapInfo PROC EXPORT

  WITH StackFrame
    link a6,*stacksize
    move.l DataSizePtr(a6),a0; compute size of parm's.
    lea  ParmEnd-ParmBeg,a1; put it in a 'c' var
    move.l a1,(a0)
    move.l TrapSizePtr(a6),a0; compute size of trap
    lea  TrapEnd-TrapBeg,a1; put it in a 'c' var
    move.l a1,(a0) ; put hook address ...
    move.l TrapAddrPtr(a6),a0
    lea  TrapBeg,a1 ; in a 'c' variable.
    move.l a1,(a0)
    unlk a6
    rts
  ENDWITH

  ParmBeg
  SvRTS    dc.l 0 ; place holder for compiling.
  FcAdr     dc.l 0 ; place holder for compiling.
  TpAdr     dc.l 0 ; place holder for compiling.
  CLv1     dc.l 0 ; place holder for compiling.
  ParmEnd

; The following block of code actually gets copied down into

```

; the SysHeap due to trap table nearness requirements...

```

TrapBeg  lea  CLv1,a0; keep track of nested hook calls.
         addq.w #1,(a0); see how many calls deep we are.
         cmpi.w #1,(a0) ; just call trap if call level > 1.
         bgt.s CallTrap

CallHook  lea  SvRTS,a0 ; save address to get back too...
         move.l (a7)+,(a0); save 'len' parm to 'DrawText'
         move.w (a7),d0 ; 'clr.w' trans'offset' to 'int'
         clr.w (a7) ; transform 'len' parm to MPW-C 'int'.
         ext.l d0 ; put 'len' parm back on stack.
         move.l d0,-(a7) ; get out address of our trap.
         move.l FcAdr,a0 ; call with MPW-C stack parms.
         jsr  (a0) ; do a 'pascal' clean up.
         add.l #12,a7
         lea  CLv1,a0; indicate exit of nested call.
         subi.w #1,(a0) ; setup jump to original caller.
         move.l SvRTS,a0 ; get out of here!!
         jmp (a0)

CallTrap  lea  CLv1,a0; indicate exit of nested call.
         subi.w #1,(a0) ; just call standard trap.
         move.l TpAdr,a0 ; this won't come back here.
         jmp (a0)

TrapEnd

        ENDPROC
; =====

GetResTrapInfo PROC EXPORT

  WITH StackFrame
    link a6,*stacksize
    move.l DataSizePtr(a6),a0; compute size of parm's.
    lea  ParmEnd-ParmBeg,a1; put it in a 'c' var
    move.l a1,(a0)
    move.l TrapSizePtr(a6),a0; compute size of trap
    lea  TrapEnd-TrapBeg,a1; put it in a 'c' var
    move.l a1,(a0)
    move.l TrapAddrPtr(a6),a0; put hook address ...
    lea  TrapBeg,a1 ; in a 'c' variable.
    move.l a1,(a0)
    unlk a6
    rts
  ENDWITH

  ParmBeg
  SvRTS    dc.l 0 ; place holder for compiling.
  FcAdr     dc.l 0 ; place holder for compiling.
  TpAdr     dc.l 0 ; place holder for compiling.
  CLv1     dc.l 0 ; place holder for compiling.
  ParmEnd

  FontID    EQU 4

; The following block of code actually gets copied down into
; the SysHeap due to trap table nearness requirements...

TrapBeg  lea  CLv1,a0; keep track of nested hook calls.
         addi.w #1,(a0) ; see how many calls deep we are.
         cmpi.w #1,(a0); just call trap if call level > 1.
         bgt.s CallTrap

CallHook: ; unload the resType from the stack.
         move.l 6(a7),d0 ; check it for type 'FOND'.
         cmpi.l #'FOND',d0 ; do nothing if it isn't.
         bne.s CallTrap; else - clear special super/sub bits
         ; so FontMgr gets Font info of a known Font.
         andi.w #$1FFF,FontID(a7)

CallTrap: lea  CLv1,a0; indicate exit of nested call.
         subi.w #1,(a0) ; just call standard trap.
         move.l TpAdr,a0; this won't come back here.
         jmp (a0)

TrapEnd

        ENDPROC

```

```
; =====
UnChainHookPROC EXPORT
    WITH StackFrame
        link    a6,#stacksize
        move.l  TrapSizePtr(a6),a0; compute size of trap
        lea     TrapEnd-NormTrap,a1; put it in a 'c' var
        move.l  a1,(a0)
        move.l  TrapAddrPtr(a6),a0; put hook address ...
        lea     NormTrap,a1; put it in a 'c' variable.
        move.l  a1,(a0)
        unlk a6
        rts
    ENDWITH

SvRTS    dc.l 0; place holder for compiling.
FcAdr    dc.l 0; place holder for compiling.
TpAdr    dc.l 0; place holder for compiling.
CLvl     dc.l 0; place holder for compiling.
```

```
NormTrap; just call standard trap.
        move.l TpAdr,a0; this won't come back here.
        jmp (a0)

TrapEnd
    ENDPROC
END
```

Listing: TrapHook.h

```
/* =====
TrapHook.h
===== */

#define SUPER 0x4000
#define SUB 0x2000
#define REGULAR 0x1FFF

typedef
struct TrapHookRecord
{
    long SvRTS;
    long FuncAddress;
    long TrapAddress;
    long HookCallLevel;
    long Code;
} TrapHookRecord, *TrapHookPointer;
```

Listing: TrapHook.c

```
/* =====
TrapHook.c
===== */

#include "TrapHook.h"
#include <Memory.h>
#include <Menus.h>
#include <strings.h>

#define DrawCharNum 0x083

extern long InsertHookAddr;
/* ===== */

SetupHook(InfoFunc,HookFunc,
    TrapNum,HookAddr,NormalTrap)
{
    int (*InfoFunc)();
    long HookFunc;
    int TrapNum;
    TrapHookPointer *HookAddr;
    long *NormalTrap;

    int ParamsSize;
    int CodeSize;
    long CodeAddr;
    long OldZone;
```

```
(*InfoFunc)(&CodeAddr,&CodeSize,&ParamsSize);

/* MAKE SURE TRAP HOOK IS IN SYS HEAP... */
OldZone = (long)GetZone();
SetZone (SystemZone());
*HookAddr = (TrapHookPointer)
    NewPtr(ParamsSize + CodeSize);
SetZone (OldZone);

*NormalTrap= GetTrapAddress(TrapNum);
(**HookAddr).FuncAddress = HookFunc;
(**HookAddr).TrapAddress = *NormalTrap;
(**HookAddr).HookCallLevel = 0;
BlockMove(CodeAddr,&(**HookAddr).Code),CodeSize);

SetTrapAddress(&(**HookAddr).Code),TrapNum);
}
/* ===== */

UnsetTrapHook(HookAddr)
{
    TrapHookPointer HookAddr;

    int CodeSize;
    long CodeAddr;

    UnChainHook(&CodeAddr,&CodeSize);

    BlockMove(CodeAddr,&HookAddr->Code),CodeSize);
}
/* ===== */
```

Listing: windows.c

```
/* =====
Windows.c
===== */

/* NEEDED TO DEFINE NEW TEXTEDIT ROUTINES IN TEXTEDIT.H */
#define __ALLNU__

#include <QuickDraw.h>
#include <Types.h>
#include <Windows.h>
#include <TextEdit.h>
#include <Controls.h>
#include <Events.h>
#include <fonts.h>
#include <toolutils.h>

#include "MiniEdit.h"

extern WindowRecord wRecord;
extern WindowPtr myWindow;
extern ControlHandle levScroll;
extern TEHandle TEH;
extern char dirty;
extern Str255 theFileName;
/* ===== */

SetUpWindows()
{
    Rect destRect, viewRect;
    Rect vScrollRect;
    FontInfo myInfo;
    int height;
    TextStyle Styl;

    myWindow = GetNewWindow(windowID,&wRecord, (WindowPtr)-1L);
    SetPort(myWindow);

    vScrollRect = (*myWindow).portRect;
    vScrollRect.left = vScrollRect.right-15;
    vScrollRect.right += 1;
```



```

vScrollRect.bottom -= 14;
vScrollRect.top    -= 1;
vScroll = NewControl(myWindow, &vScrollRect,
    "", 1, 0, 0, 0, scrollBarProc, 0L);

viewRect      = qd.thePort->portRect;
viewRect.right -= SBarWidth;
viewRect.bottom -= SBarWidth;
InsetRect(&viewRect, 4, 4);

TEH = TESTylNew( &viewRect, &viewRect );
Styl.tsFont= newYork;
Styl.tsSize= 14;
TESetStyle(doSize+doFont,&Styl,1,TEH);

SetView(qd.thePort);
dirty = 0;
}
/* ----- */

AdjustText ()
{
    int    oldScroll, newScroll, delta;

    oldScroll = (**TEH).viewRect.top - (**TEH).destRect.top;
    newScroll = GetCtlValue(vScroll) * (**TEH).lineHeight;
    delta     = oldScroll - newScroll;

    if (delta != 0)
        TESCroll(0, delta, TEH);
}
/* ----- */

SetVScroll()
{
    register int n,lines;

    lines = linesInView(TEH,NULL);

    n = (**TEH).nLines-lines;

    if (((**TEH).teLength > 0) &&
        (((**TEH).hText)[(**TEH).teLength-1]!='\r'))
        n++;

    SetCtlMax(vScroll, n > 0 ? n : 0);
}
/* ----- */

ShowSelect()
{
    register int    topLine, bottomLine, theLine, lines;

    SetVScroll();
    AdjustText();

    lines = linesInView(TEH,NULL);

    topLine = GetCtlValue(vScroll);
    bottomLine = topLine + lines;

    if ((**TEH).selStart < (**TEH).lineStarts[topLine] ||
        (**TEH).selStart >=
            (**TEH).lineStarts[bottomLine])
    {
        for (theLine = 0;
            (**TEH).selStart >= (**TEH).lineStarts[theLine];
            theLine++);

        SetCtlValue(vScroll, theLine - lines / 2);
        AdjustText();
    }
}

```

```

/* ----- */

linesInView(TEHd1,above)
    TEHandle    TEHd1;
    int         *above;
{
    register LHHandle LHHd1;
    register LHPtr    LHptr;
    register int    nLines,before,within;
    Rect            dest,view;

    dest = (**TEHd1).destRect;
    view = (**TEHd1).viewRect;

    LHHd1 = (**GetStylHandle(TEHd1)).lhTab;
    nLines = (**TEHd1).nLines;
    before = 0;

    HLock(LHHd1);
    LHptr = *LHHd1;

    while ((dest.top < 0) && (before < nLines))
        dest.top += LHptr[before++].lhHeight;

    within = before;

    while ((dest.top < view.bottom) && (within < nLines))
        dest.top += LHptr[within++].lhHeight;
    HUnlock(LHHd1);

    if (above) *above = before;
    within -= before;

    return(within);
}
/* ----- */

SetView(w)
    WindowPtr w;
{
    (**TEH).viewRect      = w->portRect;
    (**TEH).viewRect.right -= SBarWidth;
    (**TEH).viewRect.bottom -= SBarWidth;
    InsetRect(&(**TEH).viewRect, 4, 4);

    (**TEH).destRect.right = (**TEH).viewRect.right;

    TEOallText(TEH);
}
/* ----- */

UpdateWindow(theWindow)
    WindowPtr theWindow;
{
    GrafPtr savePort;

    GetPort( &savePort );
    SetPort( theWindow );

    BeginUpdate( theWindow );
    EraseRect(&theWindow->portRect);
    DrawControls( theWindow );
    DrawGrowIcon( theWindow );
    TEUpdate( &theWindow->portRect, TEH );
    EndUpdate( theWindow );

    SetPort( savePort );
}
/* ----- */

pascal void ScrollProc(theControl, theCode)
    ControlHandle theControl;
    int            theCode;
{

```

```

int pageSize;
int scrollAmt;
int lines;

if (theCode == 0)
    return;

lines = linesInView(TEH,NULL);

pageSize = lines - 1;

switch (theCode)
{
    case inUpButton:
        scrollAmt = -1;
        break;
    case inDownButton:
        scrollAmt = 1;
        break;
    case inPageUp:
        scrollAmt = -pageSize;
        break;
    case inPageDown:
        scrollAmt = pageSize;
        break;
}

SetCtlValue( theControl, GetCtlValue(theControl)+scrollAmt
);
AdjustText();
}
/* ----- */

DoContent(theWindow, theEvent)
WindowPtr theWindow;
EventRecord*theEvent;
{
    int cntlCode;
    ControlHandle theControl;
    int pageSize;
    GrafPtr savePort;

    GetPort(&savePort);
    SetPort(theWindow);
    GlobalToLocal( &theEvent->where );

    if ((cntlCode = FindControl(&theEvent->where,
        theWindow, &theControl)) == 0)
    {
        if (PtInRect( &theEvent->where, &(**TEH).viewRect ))
            TEClick( &theEvent->where,
                (theEvent->modifiers & shiftKey )!=0,
                TEH);
    }
    else
        if (cntlCode == inThumb)
        {
            TrackControl(theControl, &theEvent->where, 0L);
            AdjustText();
        }
        else
            TrackControl(theControl,
                &theEvent->where, ScrollProc);

    SetPort(savePort);
}
/* ----- */

MyGrowWindow( w, p )
WindowPtr w;
Point p;
{
    GrafPtr savePort;

```

```

long theResult;
int oScroll,b,r;
Rect rct, oView;

GetPort( &savePort );
SetPort( w );

b = qd.screenBits.bounds.bottom;
r = qd.screenBits.bounds.right;

SetRect(&rct, 80, 80, r, b);
theResult = GrowWindow( w, p, &rct );

if (theResult == 0)
    return;

SizeWindow( w, LoWord(theResult), HiWord(theResult), 1);

InvalRect(&w->portRect);
oView = (**TEH).viewRect;
oScroll = GetCtlValue(vScroll);

SetView(w);
HidePen();
MoveControl(vScroll,
    w->portRect.right - SBarWidth,w->portRect.top-1);
SizeControl(vScroll, SBarWidth+1,w->portRect.bottom -
    w->portRect.top-(SBarWidth-2));
ShowPen();

SetVScroll();
AdjustText();

SetPort( savePort );
}
/* ----- */

CloseMyWindow()
{
    HideWindow( myWindow );
    TETeSetSelect( 0, (**TEH).teLength, TEH );
    TEDelete( TEH );
    SetVScroll();
}
/* ----- */

```

Listing: SuperSub.c

```

/*****
SuperSub.c
*****/

#include <QuickDraw.h>
#include <Types.h>
#include <Fonts.h>
#include <Windows.h>
#include <Menus.h>
#include <TextEdit.h>
#include <Dialogs.h>
#include <Events.h>
#include <Desk.h>
#include <Files.h>
#include <ToolUtils.h>
#include <Controls.h>
#include <Strings.h>

#include "traphook.h"

extern TEHandle TEH;
/* ----- */

DrawTextHook(cnt,pos,text)
int cnt;
int pos;

```

```

char *text;
{
    register int fnt,offset;
    GrafPtr    gptr;

    GetPort(&gptr);
    fnt = gptr->txFont;

    offset = ((fnt & SUPER) ? -3 : ((fnt & SUB) ? 3 : 0));

    if (offset)
    { Move(0,offset);
      TextFont(fnt & REGULAR);
    }

    DrawText(text,pos,cnt);

    if (offset)
    { Move(0,-offset);
      TextFont(fnt);
    }
}
/* ----- */

```

Listing: MiniEdit.h

```

/*****
MiniEdit.h
*****/

#define windowID    128
#define ErrorAlert  256
#define AdviseAlert 257

/* resource IDs of menus */
#define appleID     128
#define fileID      129
#define editID      130

/* Edit menu command indices */
#define cutCommand  1
#define copyCommand 2
#define pasteCommand 3
#define TimesCommand 5
#define GenevaCommand 6
#define NewYorkCommand 7
#define s9Command   9
#define s10Command  10
#define s12Command  11
#define s14Command  12
#define s18Command  13
#define plainCommand 15
#define ulineCommand 16
#define boldCommand  17
#define italicCommand 18
#define shadowCommand 19
#define SuperCommand 20
#define RegularCommand 21
#define SubCommand   22

/* Menu indices */
#define appleM      0
#define fileM       1
#define editM       2

#define fmQuit      1

#define aaSave      1
#define aaDiscard   3
#define aaCancel    3

#define SBarWidth   15
#define NULL        0L

```

Listing: MiniEdit.c

```

/*****
MiniEdit.c
*****/

/* NEEDED TO DEFINE NEW TEXTEDIT ROUTINES IN TEXTEDIT.H */
#define __ALLNU__

#include <QuickDraw.h>
#include <Types.h>
#include <Fonts.h>
#include <Windows.h>
#include <Menus.h>
#include <TextEdit.h>
#include <Dialogs.h>
#include <Events.h>
#include <Desk.h>
#include <Files.h>
#include <ToolUtils.h>
#include <Controls.h>
#include <resources.h>
#include <strings.h>
#include <Scrap.h>

#include "MiniEdit.h"
#include "TrapHook.h"

#define DrawCharNum 0x085
#define GetRes      0x1A0

WindowRecord    wRecord;
WindowPtr       myWindow;
TEHandle        TEH;
Rect            dragRect = ( 0, 0, 1024, 1024 );
MenuHandle      myMenus[3];
ControlHandle    vScroll;
Cursor          editCursor;
Cursor          waitCursor;
char            dirty;
EventRecord      myEvent;
int              baseoffset = 0;
TrapHookPointer InsertHookAddr;
TrapHookPointer ResHook;
long            NormalTrap;
long            ResTrap;

GetDrawTrapInfo();
GetResTrapInfo();
DrawTextHook();
/* ----- */

main()
{
    int    myRsrc;

    InitGraf(&qd.thePort);
    InitFonts();
    FlushEvents( everyEvent, 0 );
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs(0L);
    InitCursor();
    MaxApplZone();

    SetUpCursors();
    SetUpMenus();
    SetUpWindows();

    SetupHook(GetDrawTrapInfo,DrawTextHook,
              DrawCharNum,&InsertHookAddr,&NormalTrap);
    SetupHook(GetResTrapInfo,NULL,

```

```

    GetRes,&ResHook,&ResTrap);

    while (MainEvent());

    UnsetTrapHook(InsertHookAddr);
    UnsetTrapHook(ResHook);
}
/* ----- */

doMouse(myEvent)
    EventRecord*myEvent;
{
    WindowPtr    whichWindow;

    switch (FindWindow( &myEvent->where, &whichWindow ))
    {
        case inDesk:
            break;
        case inGoAway:
            if (ours(whichWindow))
                if (TrackGoAway( myWindow,
                                myEvent->where ))
                    /*DoFile(fmClose) */
                ;
            break;
        case inMenuBar:
            return( DoCommand(
                MenuSelect(myEvent->where) ));
        case inSysWindow:
            SystemClick( &myEvent, whichWindow );
            break;
        case inDrag:
            if (ours(whichWindow))
                DragWindow( whichWindow,
                            &myEvent->where, &dragRect );
            break;
        case inGrow:
            if (ours(whichWindow))
                MyGrowWindow( whichWindow,
                              &myEvent->where );
            break;
        case inContent:
            if (whichWindow != FrontWindow())
                SelectWindow(whichWindow);
            else
                if (ours(whichWindow))
                    DoContent(whichWindow, &myEvent);
            break;
        default: ;
    }

    return(1);
}
/* ----- */

doKey(myEvent)
    EventRecord *myEvent;
{
    register char theChar;

    theChar = myEvent->message & charCodeMask;
    if ((myEvent->modifiers & cmdKey) != 0)
        return( DoCommand( MenuKey( theChar ) ));
    else
    {
        TEKey( theChar, TEH );
        ShowSelect();
        dirty = 1;
    }

    return(1);
}
/* ----- */

```

```

doActivate(myEvent)
    EventRecord *myEvent;
{
    Rect    rct;

    if (ours((WindowPtr)myEvent->message))
    {
        rct    = (*myWindow).portRect;
        rct.top    = rct.bottom - (SBarWidth+1);
        rct.left   = rct.left - (SBarWidth+1);
        InvalRect(&rct);

        if ( myEvent->modifiers & activeFlag )
        {
            TEActivate( TEH );
            ShowControl( vScroll );
            TEFromScrap();
        }
        else
        {
            TDeactivate(TEH);
            HideControl( vScroll );
            ZeroScrap();
            TToScrap();
        }
    }

    return(1);
}
/* ----- */

int MainEvent()
{
    WindowPtr    whichWindow;

    MaintainCursor();
    MaintainMenus();

    SetTrapAddress(NormalTrap,DrawCharNum);
    SetTrapAddress(ResTrap,GetRes);

    SystemTask();

    SetTrapAddress(&(InsertHookAddr->Code),
        DrawCharNum);
    SetTrapAddress(&(ResHook->Code),GetRes);

    TEIdle(TEH);
    if (GetNextEvent(everyEvent, &myEvent))
    {
        switch (myEvent.what)
        {
            case mouseDown:
                return(doMouse(&myEvent));
                break;
            case keyDown:
            case autoKey:
                return(doKey(&myEvent));
                break;
            case activateEvt:
                return(doActivate(&myEvent));
                break;
            case updateEvt:
                if (ours((WindowPtr)myEvent.message))
                    UpdateWindow(myWindow);
                break;
            default: ;
        }
    }

    return(1);
}
/* ----- */

SetUpMenus()

```

```

{
    int i;

    myMenus[appleM] = NewMenu( appleID, "\024" );
    AddResMenu( myMenus[appleM], 'DRVR' );

    myMenus[fileM] = GetMenu(fileID);
    myMenus[editM] = GetMenu(editID);

    for ( (i=appleM); (i<=editM); i++ ) InsertMenu(myMenus[i],
0);

    DrawMenuBar();
}
/* ----- */

doEditing(theItem)
int theItem;
{
    WindowPeek wPtr;
    TextStyle Styl;
    int lhgt, ascent;

    wPtr = (WindowPeek) FrontWindow();

    switch (theItem)
    { case cutCommand:
        TECut( TEH );
        dirty = 1;
        break;
      case copyCommand:
        TECopy( TEH );
        break;
      case pasteCommand:
        TEPaste( TEH );
        dirty = 1;
        break;

      case TimesCommand:
        Styl.tsFont = times;
        TETSetStyle(doFont, &Styl, 1, TEH);
        break;
      case GenevaCommand:
        Styl.tsFont = geneva;
        TETSetStyle(doFont, &Styl, 1, TEH);
        break;
      case NewYorkCommand:
        Styl.tsFont = newYork;
        TETSetStyle(doFont, &Styl, 1, TEH);
        break;

      case s9Command:
        Styl.tsSize = 9;
        TETSetStyle(doSize, &Styl, 1, TEH);
        break;
      case s10Command:
        Styl.tsSize = 10;
        TETSetStyle(doSize, &Styl, 1, TEH);
        break;
      case s12Command:
        Styl.tsSize = 12;
        TETSetStyle(doSize, &Styl, 1, TEH);
        break;
      case s14Command:
        Styl.tsSize = 14;
        TETSetStyle(doSize, &Styl, 1, TEH);
        break;
      case s18Command:
        Styl.tsSize = 18;
        TETSetStyle(doSize, &Styl, 1, TEH);
        break;
      case s24Command:
        Styl.tsSize = 24;
        TETSetStyle(doSize, &Styl, 1, TEH);
        break;

      case plainCommand:
        Styl.tsFace = 0;

```

```

        TETSetStyle(doFace, &Styl, 1, TEH);
        break;
      case boldCommand:
        TETGetStyle(**TEH).se1Start,
        &Styl, &lhgt, &ascent, TEH);
        Styl.tsFace |= bold;
        TETSetStyle(doFace, &Styl, 1, TEH);
        break;
      case underlineCommand:
        TETGetStyle(**TEH).se1Start,
        &Styl, &lhgt, &ascent, TEH);
        Styl.tsFace |= underline;
        TETSetStyle(doFace, &Styl, 1, TEH);
        break;
      case italicCommand:
        TETGetStyle(**TEH).se1Start,
        &Styl, &lhgt, &ascent, TEH);
        Styl.tsFace |= italic;
        TETSetStyle(doFace, &Styl, 1, TEH);
        break;
      case shadowCommand:
        TETGetStyle(**TEH).se1Start,
        &Styl, &lhgt, &ascent, TEH);
        Styl.tsFace |= shadow;
        TETSetStyle(doFace, &Styl, 1, TEH);
        break;

      case SuperCommand:
        TETGetStyle(**TEH).se1Start-1,
        &Styl, &lhgt, &ascent, TEH);
        Styl.tsFont &= REGULAR;
        Styl.tsFont |= SUPER;
        TETSetStyle(doFont, &Styl, 1, TEH);
        break;
      case RegularCommand:
        TETGetStyle(**TEH).se1Start-1,
        &Styl, &lhgt, &ascent, TEH);
        Styl.tsFont &= REGULAR;
        TETSetStyle(doFont, &Styl, 1, TEH);
        break;
      case SubCommand:
        TETGetStyle(**TEH).se1Start-1,
        &Styl, &lhgt, &ascent, TEH);
        Styl.tsFont &= REGULAR;
        Styl.tsFont |= SUB;
        TETSetStyle(doFont, &Styl, 1, TEH);
        break;

      default: ;
    }
    ShowSelect();
}
/* ----- */

int DoCommand( mResult )
long mResult;
{
    int theItem;
    Str255 name;

    theItem = LoWord( mResult );
    switch (HiWord(mResult))
    { case appleID:
        GetItem(myMenus[appleM], theItem, &name);
        OpenDeskAcc( &name );
        SetPort( myWindow );
        break;
      case fileID:
        if (theItem == fmQuit) return(0);
        break;
      case editID:
        if (SystemEdit(theItem-1)==0)
            doEditing(theItem);
        break;

```

```

    }
    HiliteMenu(0);
    return(1);
}
/* ----- */

MaintainCursor()
{
    Point    pt;
    WindowPeek wPtr;
    GrafPtr  savePort;
    Rect     TERect;

    wPtr = (WindowPtr)(WindowPeek)FrontWindow();

    if (ours(wPtr))
    {
        GetPort( &savePort );
        SetPort( (GrafPtr)wPtr );

        GetMouse(&pt);
        TERect = (**TEH).viewRect;

        if ( PtInRect(&pt, &TERect) )
            SetCursor( &editCursor );
        else
            InitCursor();

        SetPort( savePort );
    }
}
/* ----- */

MaintainMenus()
{
    if ( !(*((WindowPeek)myWindow).visible ||
        !ours(FrontWindow()) ) )
    {

```

```

        EnableItem( myMenus[editM], cutCommand );
        EnableItem( myMenus[editM], copyCommand );
    }
    else
    {
        if ((*TEH).selStart==(**TEH).selEnd) {
            DisableItem( myMenus[editM], cutCommand );
            DisableItem( myMenus[editM], copyCommand );
        }
        else
        {
            EnableItem( myMenus[editM], cutCommand );
            EnableItem( myMenus[editM], copyCommand );
        }
    }
}
/* ----- */

SetUpCursors()
{
    CursHandle hCurs;

    hCurs = GetCursor(1);
    editCursor = **hCurs;
    hCurs = GetCursor(watchCursor);
    waitCursor = **hCurs;
}
/* ----- */

ours(w)
    WindowPtr w;
{
    return( (myWindow!=NULL) && (w==myWindow) );
}
/* ----- */

```



C Workshop

Extend Your Favorite Editor



Joel McNamara
Bellevue, WA
Volume 5, Number 7

Extending Your Editor Edit Extender DA

If you've programmed for awhile, you're bound to to encounter one or two text editors you really feel comfortable with. Be it EMACS on a mainframe or QUED on a Mac, a good editor has a way of growing on you. You savor the power, the flexibility, and all of those little features that really make it stand out. Then comes the day, for some reason or another, when you're forced to use another editor. You click on the menu bar and your favorite feature isn't there. A keyboard command just puts a non-ASCII character on the screen. You grit your teeth and code away, wishing your preferred editor was around.

I was rudely awakened to this predicament when I started to use MPW and Lightspeed C. QUED had long been my editor of choice on the Mac, and I suddenly found a few of my favorite features weren't around anymore. Granted, I could continue to use QUED or even be more ambitious and write my own editor with a little help from Symantec's CAPPS, but I really like the notion of an integrated programming environment. To me, it's a hassle to transfer from your editor into your main development environment all of the time.

Ah, but that's the joy of being a programmer. You're not forever an end user, hoping and praying the software gods will deliver you from want. You can rise above the masses with an idea and a compiler. The problem of creating an extendible editor can easily be solved by simply writing a desk accessory.

The theory goes like this. You have a chunk of text you want to manipulate within your editor. You just select the text, copy it into the clipboard, capitalize/count/do whatever to it, and then paste the altered text back in, replacing the previous selection. You let the text editor do all of the work, while you sit on top, adding your own features through the DA. No sweat, DA spell checkers like Thunder! do it all the time. Just combine the concepts of the sample "Windows" desk accessory included with Lightspeed C with some clipboard related code, and viola, a few quick hours of programming gives you the features of your favorite editor in a least favored one. Specifically, in the following example, capitalization, changing case, word counting, time and date insertion, and saving a selected region to a text file.

The Clipboard, Fake Keydowns, and Timing

The obvious storage location for storing text to be manipulated is the clipboard, alias the desk scrap. Once the text is there, it can be twiddled with to your heart's content. So, the first thing to do is get the selected text into it. Now being somewhat lazy in not wanting to menu select Copy or Paste or type their keyboard equivalents all of the time, I decided there must be a better way,

more in tune with the Mac interface. Something along the lines of having the text selected, pulling down the DA's menu, and having the command executed.

The solution is to post a Command "C" (the non-case sensitive, universal Mac key command for "Copy"), and fake the editor into thinking the user had just done a copy into the clipboard. It's a pretty easy task to post a keydown event with the PostEvent trap, but unfortunately you don't have any control over the modifiers. If you pass in an ASCII "c" eventCode, and a "keyDwnEvt" eventMsg, all you'll get is a lower case "c" on the screen. There are no provisions for specifying that the command/option/shift/caps lock keys may also be pressed. There are two choices in dealing with this problem: (1) Post some mousedown events with the locations of the "Edit" menu and "Copy" position, essentially faking a menu selection with the mouse. A risky proposition since you rely on the editor following Apple's interface guidelines of where things should be. (2) Fake the Mac into thinking a command modifier key has been pressed.

The quick and dirty way is to pick door number 2 and modify the event queue. Using TMON, the location of the key modifiers are readily apparent. (Snoop around at \$174 and start pressing keys.) Just change a byte here and there, and presto, the keydown "c" event is turned into a Command "C" copy event. The same technique can be used when replacing text, by posting a Command "V" for paste. You just select a menu item, Edit Extender posts a Command "C," and the text is sitting in the clipboard ready for manipulation.

At this point, things begin to get a bit murky. If you were to implement the above method in a desk accessory, you'd see the menu bar flash, signalling the Command "C" had been posted, but no text would be copied into the clipboard. What's going on here? The solution is a matter of timing. The DA has to tell the operating system that it's finished, have the application take control again to do the copy, and then signal it wants to take control again. This can be accomplished with the desk accessory's csCode parameter. Just pass in an integer in the accRun message and branch as required. With the timing problem solved, Edit Extender worked great. I now had some of my favorite features up and running. Then I was in for a rude surprise. Out of curiosity, I decided to try running it with MacWrite and Microsoft Word. It didn't work at all. Something strange was going on. The DA worked fine with MPW, Lightspeed C, miniEdit, and QUED, but fell flat on its face when it came to "real" word processors.

Private Scraps

Despite having the same purpose, when it comes to using the clipboard and copying and pasting, text editors and word proces-

sors are two different critters. Most word processors that do any kind of text formatting (bolding, underlining, italicizing, etc.) keep the text they are dealing with in what "Inside Macintosh" refers to as a "private scrap." This is simply a private clipboard where the application stores text and formatting information for internal use. For the sake of speed and efficiency, when the user selects some text and copies or cuts it, more than likely it's going to end up someplace other than the desk scrap. On the other hand, virtually every text editor that deals strictly with text, will use the desk scrap for storage. This is why you can access copied text through the clipboard from the Lightspeed or MPW editor, but not from MacWrite or Word.

The Scrap Manager section of "Inside Macintosh" enlightens you to the fact that it is good and required programming etiquette for an application to transfer its private scrap to the common access desk scrap whenever a user quits the program or when a desk accessory becomes active. In doing so, formatted data should be converted into a form readable by either another application or the desk accessory. That means text turns it into a type 'TEXT' resource (simply a series of ASCII characters), graphics become a 'PICT' type, and optionally, any formatted information can be kept with its own unique resource type.

I fought off the temptation to cop-out and use Edit Extender exclusively with text editors. After all, it might be nice to have it work with word processors or even desktop publishing applications. So, what determines if a desk accessory is activated or not? Simple, if a window that is owned by the DA is brought to the front and becomes the active window, the desk accessory becomes activated. If another non-DA window comes to the front, the DA is deactivated. Not wanting to splash an ugly window on the screen, I decided to make my window appear offscreen at the upper left corner. Now when Edit Extender got a command, it brought the offscreen window to the front, the application received a message that a DA had become active and converted its private scrap into the desk scrap. The DA goes away, and presto, text now awaits us in the desk scrap.

With the window code working, Edit Extender was in business with MacWrite. With the exception of one small detail. If you had some formatted text (say italicized Geneva 10) selected, and used Edit Extender to capitalize it, when it was pasted back in, it came back as Geneva 12 plain. Clearly a case of WYDWISYG (What you don't want is what you get!). Remember that when the private scrap is converted, it can also place formatted data in the desk scrap. That's exactly what MacWrite is doing. In addition to the 'TEXT' resource, MacWrite is placing a 'MWRT' resource in the desk scrap that contains the text with formatting information. When MacWrite pastes back in, it will first look in the scrap for a 'MWRT' resource, and if it's there, paste it in with all formatting intact. If there's none to be found, 'TEXT' will be pasted in at Geneva 12. The solution to this problem is to get the clipboard format of the 'MWRT' text. In this case, look at Tech Note #13, MacWrite Clipboard Format. Although this version of Edit Extender doesn't support formatted text, to do so would be fairly trivial. After you've posted the copy event, just scan the desk scrap for

a resource with the type 'MWRT.' If you find it, then based on the data structure, go in and find the text and manipulate it.

Microsoft machinations

Hmmm. Now it was working with MacWrite, but wasn't with Word. Ack! The simple DA was turning into a frustrating programming exercise. Oh well, when in doubt, get your debugger out. TMON revealed an interesting characteristic in Word. In addition to requiring the desk accessory to be activated, it also needed a Copy (Command "C") or Paste (Command "V") event before it would convert its private scrap to the desk scrap. Picky, picky, picky. After sliding in the appropriate code, and seeing no problems with using it for other applications, Edit Extender now seemed to work just fine. (Be advised though, that unlike MacWrite, the current version 3.0_whatever doesn't place any formatted text into the clipboard. Rumor has it that future versions will place RTF into the desk scrap along with unformatted text.)

MultiFinder considerations

With any desk accessory that interacts directly with an application, you need to keep in mind MultiFinder and its buddy the DA Handler. Since Mr. DA Handler treats a desk accessory like an individual program, I was expecting some major complications. I know, Apple says to write small applications instead of DAs, but I still like interactive desk accessories. Fortunately in their infinite wisdom, the creators of MultiFinder left a nice loophole for DAs like Edit Extender. The solution is to hold down the Option key when initially selecting and launching the desk accessory. That means click on the apple menu, hold down the Option key, and select the DA. (If you hold down the option key before you select the apple menu, MultiFinder won't display any DAs.) This little known technique bypasses the DA Handler, and installs Edit Extender (or any other DA for that matter) directly into the application's heap.

(MultiFinder Caveats: In using the Option key method be aware of two things. (1) If you get out of memory errors, quit the program and increase the application's memory size. (2) This technique allows you to run the desk accessory within only one application at a time. If you wanted to run Edit Extender in (i.e) MacWrite and PageMaker at the same time, just create a duplicate of Edit Extender and use ResEdit to give it a different DRVr resource name. Font DA/Mover will give it a unique resource I.D. when you install it, and you'll be set.)

Conclusion

Edit Extender was built so I could have some features available no matter which Mac editor I had to use. It demonstrates some neat tricks like posting copy and paste events and converting text for DA use. But for the most part, it's pretty much of a base to work off of. I included my pet features, and with a bit of coding, so can you. Some other options that could easily be incorporated include: sorting, search and replace, paragraph numbering, selected text printing, or indexing. Happy editor constructing!

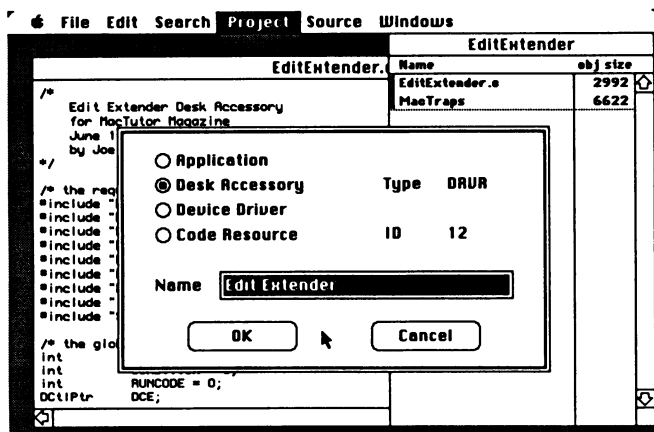


Figure 1. Project

Listing EditExtender.c

```

/*
  Edit Extender Desk Accessory
  for MacTutor Magazine
  June 1, 1988
  by Joel McNamara, Satori Software
*/
/* the required stuff... */
#include "EventMgr.h"
#include "MenuMgr.h"
#include "MemoryMgr.h"
#include "FontMgr.h"
#include "FileMgr.h"
#include "DeviceMgr.h"
#include "DialogMgr.h"
#include "Int1Pkg.h"
#include "StdFilePkg.h"

/* the globals... */
int ALREADY_OPEN = 0;
int CONDITION = 0;
int RUNCODE = 0;
DctlPtr DCE;
GrafPtr SAVEPORT;
MenuHandle THE_MENUHANDLE;

/* the constants... */
/* routine returns correct resource number for the driver */
#define RsrcID(id) (0xC000 + (~DCE->dCtlRefNum << 5) + id)
/* the dialogs */
#define AboutDLOG 0
#define CountsDLOG 1
#define MessageDLOG 2
/* counts DITL items */
#define CharItem 2
#define WordsItem 3
#define SentenceItem 4
#define ParagraphItem 5
/* the fake window */
#define FakeWIND 0
/* run code commands */
#define UpperCaseCOND 1
#define LowerCaseCOND 2
#define ReverseCaseCOND 3
#define CapitalizeCOND 4
#define CountsCOND 5
#define TextCOND 6
#define DateCOND 7
#define TimeCOND 8
/* STR# error messages */
#define ListSTR 0

```

```

#define MemoryMSG 1
#define GenericMSG 2
#define CreateMSG 3
#define OpenMSG 4
#define WriteMSG 5
#define SelectMSG 6
/* menu items */
#define AboutITEM 1
#define UpperITEM 3
#define LowerITEM 4
#define ReverseITEM 5
#define CapitalITEM 6
#define DateITEM 8
#define TimeITEM 9
#define CountsITEM 11
#define TextITEM 13
#define QuitITEM 15
/* DA messages */
#define OpenMessage 0
#define ControlMessage 2
#define CloseMessage 4

/* let's start... */
main(controlParam, dControl, message)
cntrlParam *controlParam;
DctlPtr dControl;
short message;
{
    EvQE1 tempQElement;

    if ((dControl->dCtlStorage) == 0) {
        SysBeep(5);
        if (message == OpenMessage) {
            SysBeep(5);
            Display(MemoryMSG);
            CloseDriver(dControl->dCtlRefNum);
        }
        return(0);
    }
    DCE = dControl;
    switch (message) {
        case OpenMessage:
            DoOpen();
            break;

        case ControlMessage:
            switch (controlParam->csCode) {
                /* who needs to be in control */
                case accRun:
                    if ((RUNCODE != 0) && (OSEventAvail(keyDownMask,
                        &tempQElement) == 0))
                        DoRuncode(RUNCODE);
                    break;

                case accMenu:
                    DoMenu(controlParam->csParam[1]);
                    break;

                case goodBye:
                    DoGoodBye();
                    break;
            }
            break;

        case CloseMessage:
            DoClose();
            break;
    }
    return(0);
}

/* open the DA */
DoOpen()
{

```

```

DCE->dCtlFlags |= dNeedLock|
dNeedGoodBye|dCtlEnable|dNeedTime;
DCE->dCtlDelay = 15;
DCE->dCtlMenu = RsrcID(0);
if (ALREADY_OPEN)
    return;

RUNCODE = 0;
ALREADY_OPEN = 1;
THE_MENUHANDLE = GetMenu( DCE->dCtlMenu);
(*THE_MENUHANDLE)->menuID = DCE->dCtlMenu;
InsertMenu(THE_MENUHANDLE,0);
DrawMenuBar();
}

/* handle any menu items */
DoMenu(theItem)
int theItem;
{
    switch (theItem)
    {
        case AboutITEM:
            DoAbout();
            break;

        case UpperITEM:
            DoCopy();
            CONDITION = UpperCaseCOND;
            break;

        case LowerITEM:
            DoCopy();
            CONDITION = LowerCaseCOND;
            break;

        case ReverseITEM:
            DoCopy();
            CONDITION = ReverseCaseCOND;
            break;

        case CapitalITEM:
            DoCopy();
            CONDITION = CapitalizeCOND;
            break;

        case DateITEM:
            SetRunCode();
            CONDITION = DateCOND;
            break;

        case TimeITEM:
            SetRunCode();
            CONDITION = TimeCOND;
            break;

        case CountsITEM:
            DoCopy();
            CONDITION = CountsCOND;
            break;

        case TextITEM:
            DoCopy();
            CONDITION = TextCOND;
            break;

        case QuitITEM:
            DoClose();
            break;

        default:
            break;
    }
    HiliteMenu(0);
}

```

```

/* about us */
DoAbout()
{
    DialogPtr theDialog;
    int dummy;

    theDialog = GetNewDialog(RsrcID(AboutDLOG),0,-1);
    ModalDialog(0, &dummy);
    DisposDialog(theDialog);
}

/* close everything up */
DoClose()
{
    DeleteMenu(DCE->dCtlMenu);
    DisposeMenu(THE_MENUHANDLE);
    ALREADY_OPEN = 0;
    DCE->dCtlMenu = 0;
    DCE->dCtlWindow = 0;
    DrawMenuBar();
}

/* good bye... */
DoGoodBye()
{
    DCE->dCtlMenu = 0;
    DCE->dCtlWindow = 0;
}

/* do a control c */
DoCopy()
{
    struct keys{
        long charCode;
        long modifiers;
    } *keysPtr;
    long len;

    GetPort(&SAVEPORT);
    len = ZeroScrap();
    keysPtr = (struct keys*)0x174;
    keysPtr->charCode = 0x00000000;
    keysPtr->modifiers = 0x00008000;
    PostEvent(3,67);
    keysPtr->charCode = 0x00000000;
    keysPtr->modifiers = 0x00008000;
    RUNCODE = 1;
}

/* get the DA's attention */
SetRunCode()
{
    RUNCODE = 1;
}

/* handle any run codes */
DoRunCode(myRunCode)
short myRunCode;
{
    switch (myRunCode)
    {
        case 1: /* bring up the dummy window */
            DCE->dCtlWindow = GetNewWindow(RsrcID(FakeWIND),0,-1);
            ((WindowPeek)(DCE->dCtlWindow))->windowKind = DCE-
>dCtlRefNum;
            SetPort(DCE->dCtlWindow);
            RUNCODE = 2;
            break;

        case 2: /* do a paste event */
            PostPaste();
            RUNCODE = 3;
            break;
    }
}

```

```

    case 3: /* now deal with our commands */
        DoText();
        break;

    default:
        break;
}

/* here's where we do our edit commands */
DoText()
{
    long len;
    long myOffset;
    long loop;
    long temp;
    long theSecs;
    Handle myHandle;
    Handle newHandle;
    Handle theHandle;
    Rect itemRect;
    int sentences;
    int paragraphs;
    int words;
    int chars;
    int itemType;
    int tempInt;
    char myChar;
    char prevChar;
    DialogPtr theDialog;
    Ptr myPtr;
    Str255 numStr;
    Str255 timeStr;

    RUNCODE = 0;
    myHandle = NewHandle(0);

    len = GetScrap(myHandle, 'TEXT', &myOffset);
    if ((len <= 0) && (CONDITION <= CapitalizeCOND))
        /* nothing's in the scrap */
        {
            SysBeep(5);
            Display(SelectMSG);
            DisposeWindow(DCE->dCtlWindow);
            DCE->dCtlWindow = 0;
            SetPort(SAVEPORT);
        }
    else
    {
        if (CONDITION < CountsCOND)
            /* do we need a handle for manipulation? */
            newHandle = NewHandle(GetHandleSize(myHandle));

        switch (CONDITION)
        {
            /* the upper routine */
            case UpperCaseCOND:
                for (loop = 0; loop <= (GetHandleSize(myHandle));
                    loop++)
                    *(((char *) (newHandle)) + loop) = toupper(*((char *)
                        ((long)(myHandle) + loop)));
                    break;

            /* the lower case routine */
            case LowerCaseCOND:
                for (loop = 0; loop <= (GetHandleSize(myHandle));
                    loop++)
                    *(((char *) (newHandle) + loop)) =
                    tolower(*((char *) ((long)(myHandle) + loop)));
                    break;

            /* the reverse case routine */
            case ReverseCaseCOND:

```

```

                for (loop = 0; loop <= (GetHandleSize(myHandle));
                    loop++) {
                    if (isupper(*((char *) ((long)(myHandle) + loop)))
                        *(((char *) ((long)(newHandle) + loop)) =
                        tolower(*((char *) ((long)(myHandle) + loop)));
                    else
                        if (islower(*((char *) ((long)(myHandle) +
                            (long)loop)))
                            *(((char *) ((long)(newHandle) + loop)) =
                            toupper(*((char *) ((long)(myHandle) + loop)));
                        else
                            *(((char *) ((long)(newHandle) + loop)) = *((char *)
                                ((long)(myHandle) + loop));
                            }
                    break;

                /* the capitalization routine */
                case CapitalizeCOND:
                    prevChar = (char)'x';
                    for (loop = 0; loop <= (GetHandleSize(myHandle));
                        loop++)
                        {
                            if (loop == 0)
                                {
                                    *(((char *) ((long)(newHandle) + loop)) =
                                    toupper(*((char *) ((long)(myHandle) + loop)));
                                    prevChar = *(((char *) ((long)(newHandle) + loop)));
                                }
                            else
                                {
                                    if ((prevChar == ' ') || (prevChar == '\r') ||
                                        (prevChar == '\t'))
                                        *(((char *) ((long)(newHandle) + loop)) =
                                        toupper(*((char *) ((long)(myHandle) + loop)));
                                    else
                                        *(((char *) ((long)(newHandle) + loop)) =
                                        *(((char *) ((long)(myHandle) + loop)));
                                        prevChar = *(((char *) ((long)(newHandle) + loop)));
                                    }
                                }
                            break;

                /* a simplistic wc */
                case CountsCOND:
                    chars = sentences = paragraphs = words = 0;

                    myPtr = myHandle;
                    for (temp = 0; temp <= len; temp++)
                        {
                            myChar = *(((char *) ((long)myPtr + temp)));
                            chars++;

                            if (myChar == '\r')
                                paragraphs++;

                            if (myChar == '.' || myChar == '!' || myChar == '?')
                                sentences++;

                            if (myChar == ' ' || myChar == '\t' || myChar == '\r')
                                words++;
                        }

                    InitCursor();
                    theDialog = GetNewDialog(RsrcID(CountsDLOG), 0, -1);
                    SetPort(theDialog);

                    NumToString(chars - paragraphs, numStr);
                    GetDItem(theDialog, CharItem, &itemType, &theHandle, &itemRect);
                    SetIText(theHandle, numStr);
                    NumToString(words, numStr);
                    GetDItem(theDialog, WordsItem, &itemType, &theHandle, &itemRect);
                    SetIText(theHandle, numStr);
                    NumToString(sentences, numStr);

```

```

GetItem(theDialog, SentenceItem, &itemType, &theHandle, &itemRect);
SetIText(theHandle, numStr);
NumToString(paragraphs, numStr);
GetItem(theDialog, ParagraphItem, &itemType, &theHandle, &itemRect);
SetIText(theHandle, numStr);

    ModalDialog(0, &tempInt);
    CloseDialog(theDialog);
    break;

/* text to save condition */
case TextCOND:
    Extract(myHandle);
    break;

/* display date condition */
case DateCOND:
    GetDateTime(&theSecs);
    IUDateString(theSecs, longDate, &timeStr);
    PtoCstr(timeStr);
    ZeroScrap();
    PutScrap(strlen(timeStr), 'TEXT', &timeStr);
    PostPaste();
    break;

/* display time condition */
case TimeCOND:
    GetDateTime(&theSecs);
    IUTimeString(theSecs, TRUE, &timeStr);
    PtoCstr(timeStr);
    ZeroScrap();
    PutScrap(strlen(timeStr), 'TEXT', &timeStr);
    PostPaste();
    break;

default:
    break;
}

DisposeWindow(DCE->dCtlWindow);
DCE->dCtlWindow = 0;
SetPort(SAVEPORT);

if (CONDITION < CountsCOND)
/* then we need to paste in the new stuff */
{
    ZeroScrap();
    PutScrap(GetHandleSize(myHandle), 'TEXT', *newHandle);
    DisposHandle(newHandle);
    PostPaste();
}
}
DisposHandle(myHandle);
CONDITION = 0;
}

/* do a paste event */
PostPaste()
{
    struct keys{
        long charCode;
        long modifiers;
    } *keysPtr;

    keysPtr = (struct keys*)0x174;
    keysPtr->charCode = 0x00000000;
    keysPtr->modifiers = 0x00008000;
    PostEvent(3, 86);
    keysPtr->charCode = 0x00000000;
    keysPtr->modifiers = 0x00000000;
}

/* error message display */
Display(theIndex)

```

```

int theIndex;
{
    DialogPtr theDialog;
    int dummy;
    Str255 theMessage;

    GetIndString(&theMessage, RsrcID(ListSTR), theIndex);
    InitCursor();
    theDialog = GetNewDialog(RsrcID(MessageDLOG), 0, -1);
    ParamText(theMessage, "\p", "\p", "\p");
    ModalDialog(0, &dummy);
    DisposDialog(theDialog);
}

/* save selected text to a file */
Extract(myHandle)
Handle myHandle;
{
    SFReply myReply;
    Point myLoc;
    int errCode, refNum;
    long theCount;

    myLoc.v = 90;
    myLoc.h = 90;

    HLock(myHandle);
    SFPutFile(myLoc, "\pSave selected text
to:", "\pUntitled", 0L, &myReply);

    if (myReply.good)
    {
        errCode =
        Create(myReply.fName, myReply.vRefNum, 'EDIT', 'TEXT');
        if (errCode != noErr);
        {
            if (errCode == dupFNErr) {
                FSDelete(myReply.fName, myReply.vRefNum);
                errCode = Create(myReply.fName, myReply.vRefNum,
'EDIT', 'TEXT');
            }
            else if (errCode != noErr) {
                Display(CreateMSG);
                HUnlock(myHandle);
                return;
            }
        }

        if (FSOpen(myReply.fName, myReply.vRefNum, &refNum) !=
noErr) {
            Display(OpenMSG);
            FSClose(refNum);
            HUnlock(myHandle);
            return;
        }

        theCount = GetHandleSize(myHandle);
        if (FSWrite(refNum, &theCount, *myHandle) != noErr) {
            Display(WriteMSG);
            FSClose(refNum);
            HUnlock(myHandle);
            return;
        }

        FSClose(refNum);
        HUnlock(myHandle);
    }

/* just in case your compiler doesn't already have these...
otherwise, leave them out... */
strlen(s)
register char *s;
{

```

```

char *s0 = s;

while (*s++);
return (s-s0-1);
}

int toupper(c)
char c;
{
    return((c)=='a') && (c!='z') ? (c-32) : c;
}

int tolower(c)
char c;
{
    return((c)=='A') && (c!='Z') ? (c+32) : c;
}

int isupper(c)
char c;
{
    return((c)=='A') && (c!='Z');
}

int islower(c)
char c;
{
    return((c)=='a') && (c!='z');
}

```

Listing: EditExtender.r

```

* resource file for Edit Extender DA
* by Joel McNamara
* for MacTutor Magazine
* June 1, 1988

```

EditExtender.rsrc

```

TYPE DLOG
    ,-16000
    blah blah
    54 74 190 452
    Visible NoGoAway
    1
    0
    -16000

```

```

TYPE DLOG
    ,-15999
    Counts
    264 20 330 494
    Visible NoGoAway
    0
    0
    -15999

```

```

TYPE DLOG
    ,-15998
    info
    68 120 152 408
    Visible NoGoAway
    1
    0
    -15998

```

```

TYPE DITL
    ,-16000
    4
    staticText Disabled
    4 10 22 330
    Edit Extender version 1.0 - by Joel McNamara
    button

```

```

109 274 129 353
OK
staticText Disabled
112 7 129 210
For MacTutor Magazine

```

```

staticText Disabled
30 21 96 364
Edit Extender is a series of helpful tools, not found in all
text editors. Use the Edit Extender DA to change, insert, and
analyze selected text in your documents.

```

```

TYPE DITL
    ,-15999
    10
    button
    32 367 51 443
    OK

```

```

staticText Disabled
7 89 27 149
999999

```

```

staticText Disabled
32 88 52 149
999999

```

```

staticText Disabled
32 257 52 318
999999

```

```

staticText Disabled
7 257 27 318
999999

```

```

staticText Disabled
7 8 27 87
Characters:

```

```

staticText Disabled
32 8 52 56
Words:

```

```

staticText Disabled
32 171 52 246
Sentences:

```

```

staticText Disabled
7 171 27 255
Paragraphs:

```

```

staticText Disabled
6 346 25 463
Text Information

```

```

TYPE DITL
    ,-15998
    3
    button
    60 195 79 263
    OK

```

```

staticText Disabled
8 49 56 272
"0

```

```

iconItem Disabled
8 8 40 40
0

```

```

TYPE WIND
    ,-16000
    Dummy

```

```

-40 -40 -20 -20
Visible NoGoAway
0
0

```

```

TYPE MENU
    ,-16000
    §
    About Edit Extender...
    (-
    Upper case
    Lower case
    Reverse case
    Capitalize
    (-
    Current date
    Current time
    (-
    Counts...
    (-
    Save selection as...
    (-
    Quit

```

```

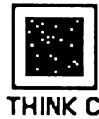
TYPE STR#
    ,-160000
    6
    Sorry, not enough memory for Edit
    Extender to run.
    Generic error!
    Sorry, couldn't create the text file.
    Sorry, couldn't open the text file you
    just created.
    Sorry, couldn't write the selected text
    to a text file.
    You must have text selected before
    performing a Edit Extender command.

```



C Objects

Pseudo Objects



Adam Treister
Santa Barbara, CA
Volume 5, Number 8

Adam Treister is the president of For Your Information in Santa Barbara, CA, and, when not spending too much time writing this article, is putting the finishing touches on Commander Bond's Briefcase, a collection of vertical market and small business calculations. He would love feedback on the article, and to have others implement more drawing features and send them to AppleLink D3474.

Evolution of the Macintosh

I recently attended one day of the Apple Worldwide Developers Conference 89, and quickly caught the thrust of Apple's unambiguous message to developers. To quote one of the slides (and Greg Williams) "If you don't learn object-oriented programming now, you will not be able to program the Macintosh later." There is not a lot of room for interpretation there. When C++ becomes available, a large number of C programmers will be switching to it. With the standardization offered by a joint development between Apple and AT&T, the nice extensions C++ adds to straight C programming, the support of Object Oriented Programming and the object libraries available through MacApp and other sources, C++ seems to be destined to become a standard Macintosh language.

From what I heard, few developers were arguing with Apple's hard line support of Object Oriented Programming. After all, the evolution of the Macintosh is looking pretty solid. The Mac is reaffirming its roots in the object oriented environment of the Xerox Star. It is striving to become a large scale operating system with virtual memory and (someday) true multi-tasking, and it is moving to a software environment that allows maximum flexibility and maintainability. And, a few years down the road, just when OS/2 is finally overcoming the obstacles of its own weight and poor design, the Macintosh, on the foundation of a (then) completely object oriented software base, will be in a position to make a seamless transition to multiprocessor architectures, and blow the blue suits out of the water.

A Gentle Transition

With these dreams as our beacon, and the dogcow cheering us onward, developers will look to C++ as a way to shift into object oriented programming without sacrificing performance. Yet there seemed to be a collective angst amongst the C programmers, that the transition to C++ will be a long and burdensome one and that the costs will be high. Personally, I don't expect the transition to be that overwhelming. There is no complete rewrite necessary, as there is when switching between two syntactically different languages. C++ includes all of C, and the compiler will not be grading you on how object oriented your code is. The transition can be gradual, at your own speed, and within your own

style. The additional features of C++ should encourage good programming style, rather than choke on any of your existing bad habits.

As part of my own transition to C++, I have written here a program in a "pseudo" object oriented style. It is meant to demonstrate that object oriented programming is as much a style of programming as a characteristic of a language. It is also meant to put my code into a style such that, when I switch to C++, I will only need to make some textual substitutions rather than to completely restructure my applications.

True Object Oriented Programming

Definitionally, an object oriented language must satisfy all of the following properties:

- data abstraction
- inheritance
- polymorphism
- dynamic binding

This program will implement the first three properties in conventional C code. I will also discuss a possible implementation of the last property, though it will not be implemented, both for the implementation problems as well as the conceptual obscurity of dynamic binding. Because of that short-coming, the occasional complete disregard for the objects when I felt I could speed up the code, and for the love of a good acronym, we cannot claim this program matches the definition of object-oriented languages. Instead, I will call it Pseudo Object Oriented Programming.

In this code are actually two entities, which may be of interest to the reader. The first is a small shell, handling the start-up, main event loop, and shutdown responsibilities, and dispatching messages to the window objects who take over from there. This POOPShell (consisting of the first two program modules) is much the same event handling shell that I use in my commercial programs. It is almost a mini- MacApp in a few pages of source code. The second part, POOPDraw is an actual application, a MacDraw clone, which almost ranks as a real program. I hope that this program may act as the starting point for your drawing programs or drawing portions of larger programs. Although POOPDraw is only the starting point for larger applications, I think there is a vast potential for adding graphical editing capabilities to a wide variety of programs, which may not ever get them if the programmer has to start from scratch. Hopefully, this program will provide a module with the necessary capabilities to let you add simple graphic editing to your applications. If this is the case (and I hope you'll let me know if it is), then we may together verify all the good things Apple is

preaching about the drop-in extensibility of object oriented programming.

First, I will give a brief description of the organization of POOPShell and POOPDraw, and then discuss how it fulfills (or does not) the criteria of object oriented programming. There are many places where I will rely on conventions or self-imposed practices to gain advantages that are inherent in true object oriented languages. But the point of this article is that the concepts of C++ and Object Oriented Programming are not really that new to most of you. You may need to learn the extra syntax of an expanded language, but the philosophy of Object Oriented Programming will probably end up looking like an elegant manifestation of all the good habits you've been trying to maintain all along.

Objects and the POOPShell

An object, as used in this program, is a handle, just like in any other Macintosh program. It is created like any other handle, through the standard routines of the Memory Manager. I use one of my favorite macros called `_GetHandleToRecord()` which takes as its argument the typedef of a structure and returns a handle to that many bytes, after checking that all went well. The function which creates the new object (poignantly called `New`) initializes the object to know its own identity, from that point on, the object is held be responsible for all messages that might be passed to it. To the program as a whole, all objects are of the generic type `ObjectHandle`, but once a message is passed to this object, it will provide the access to the private workings of its class.

The early Greeks are rarely credited for their work in Compiler Design, but if they had been into it, Archimedes would have been quoted as saying: "Give me a good jump table and I won't have to carry this 2x4 around with me all the time." The crux of this article is that one well-placed switch statement can turn C into a sufficiently object oriented language.

By convention, the first several fields in the structure of any object are reserved for those which are common to all objects. Only the first field is truly necessary to accomplish modularity of code, but I've also included a few other fields which I've arbitrarily chosen as a list of fields I think all objects should have. The list is cut down to the minimum for this article, but you can easily add your own standard fields by modifying the macro. All standard fields are conveniently included in all objects by condensing all of their declarations into a macro called `_StdObjectFields`, which all objects must include at the top of their type declaration:

```
/* The field decls common to all objects */
```

```
#define _FLD1 void(*dispatch)();
#define _FLD2 WindowPtr port;
#define _FLD3 short class;
#define _FLD4 int length;
#define _FLD5 Rect bounds;
#define _FLD6 long attributes;

#define _StdObjectFields _FLD1 _FLD2 _FLD3 \
    _FLD4 _FLD5 _FLD6
```

```
/** This macro is included in the structure definition of all
objects. For example, a hypothetical object declaration might
appear: */
```

```
typedef struct
(
    _StdObjectFields
    /* object specific fields go here */
) HypoObjRec, *HypoObjPtr, **HypoObjHandle;
```

The first field of all objects, and the one crucial to this whole implementation, is a function pointer called **dispatch**. Upon creation of a new object, we initialize this field to point to a function specific to this class of object. The function contains little more than a switch statement, which looks at the parameter called **Message**, and according to its contents, calls some other function. All of the object's routines are declared static (which I #define into **private**) and are known only to this particular class of object. This establishes a single entry point to all of the methods for any given object, much as the `ListManager` or `Packages` have a single trap and a selector field, which determines the functionality within the trap call.

The `ParmP` parameter is a pointer to a long, which means who knows what it points to. It is a wildcard pointer, which will mean different things in each function call. If the call only has one parameter, `ParmP` points to it. If there are multiple parameters, `ParmP` is an array of pointers. Surprisingly, the entire `POOPDraw` only has one function with more than one parameter, and it has two parameters, and though it is risky practice to send blind pointers around, the potential errors it causes are easily recognized and not too pesky. (Yes, a hack is a hack.)

To dispatch any message, you simply call the function

```
Dispatch(ObjectH,message,ParmP);
```

Dispatch checks that the object exists, pulls out the function pointer, and invokes it:

```
void Dispatch(ObjectH,Message,ParmP) ObjectHandle ObjectH;
int Message;
LPtr ParmP;
{ if (ObjectH)
  (*(ObjectH)>dispatch)(ObjectH,Message,ParmP); }
This will send execution into the object's local ObjDispatch
function:
```

```
ObjDispatch(ObjectH,message,inParm,outParm)
DrawPanelHandle ObjectH;
int message;
LPtr inParm,outParm;
{
    switch (message)
    {
        case DISPOSE: Dispose(ObjectH);
        break;
        case KEYDOWN: KeyDown(ObjectH);
        break;
        case MOUSEDOWN: MouseDown(ObjectH);
        break;
        case UPDATE: Update(ObjectH);
        break;

        /* more messages down here */
    }
}
```

This extra jump table actually adds little or no overhead to the program because it will eliminate many others that would otherwise be scattered throughout the code. For example, to draw a graphic element in a non-object oriented approach, you may include code of the form:

```
switch ((*ObjectH)->class)
{
    case RECT: DrawRect(element);
               break;
    case OVAL: DrawOval(element);
               break;
    case LINE: DrawLine(element);
               break;
}
```

whereas in this program, your draw routine will look like this:

```
Dispatch(element, DRAW, NULL);
```

regardless of what kind of object you are drawing.

The beauty of this approach is that when you later expand the program, for example by adding Bezier Curves as a new graphic element, you simply add all of the methods in the module with the new object, and do not have to modify the Dispose, Activate, Mousedown, Keydown, Update, etc routines of your shell to add the extra handlers. The object itself, through the function pointer in the dispatch field, will know where to call the methods it understands. For those of you who use prototypes to check parameters (and if you don't, you should!) this also means that you need only to include the prototypes in one module, and not throughout the entire project. This will save many global recompilations, when you make changes to the parameter lists of your functions. Admittedly this wanton disregard for strongly typed pointers undermines much of the effectiveness of using prototypes, but they are still valuable within object modules.

The single exception to this modularity is the New function, which is responsible for creating all objects known to the program. Since a message can't be dispatched to the function pointer of an object not yet created, the New function for any additional classes must be added to the appropriate application-specific module. True object oriented languages also handle new slightly different than other messages, but they do it more gracefully than I do here. But this implementation is better than allowing any piece of code to create new objects directly, because it localizes where the changes must be made, and limits recompilation to a single small module, instead of forcing a global recompile. It also centralizes the creation process, allowing for error checking (not that I ever do error-checking) of new objects to be performed all in the same place.

Toolbox objects which have a refCon field (Windows, Menus, Controls, etc.), should insert the objectHandle into their refCon field. This allows a smooth interaction between the toolbox calls and the POOPShell. The only such objects I use here are Windows. To facilitate their handling of Windows, I

have yet another function **WDispatch** which allows the passing of a WindowPtr instead of always having to extract the refCon. This smoothes the interface with routines like FrontWindow() and FindWindow(), which are used throughout most Macintosh programs.

A possible improvement (at least in terms of execution speed) would be to implement Dispatch and WDispatch as macros instead of functions. Because they are called so often, this change could cut the size of the stack almost in half. But, for didactic and debugging purposes, it is valuable to be able to trace through the Dispatch function. More importantly, my attempt to implement those macros introduced some nasty interactions between the calls of the function pointer and the surrounding code.

POOPDraw

I will say very little about the application itself. It is a MacDraw-clone, which is truly the prototypical object oriented program example. Each graphic element you draw is an object, as are the window, and the view, which handles to the content region of the window. The tool palette is a cheap hack, using a single picture resource as the whole palette, instead of fifteen separate objects each with their own pictures and event handlers. It is actually implemented this way intentionally, as an example of when it's easier to just cheat than to create real objects. A better job would use Thomas Fruin's tear-off windowing techniques (MacTutor, 12/88), making TWindow objects.

In the last minute scramble, to finish a long overdue project, I am axing many expected features, which any real application must have. Major examples include printing, scrolling and the entire Edit menu. We'll leave that stuff as exercises for the reader. I did implement a basic file I/O scheme, because that's usually omitted from this type of article and is great code to steal if you've never done it or written it yourself. I thought it would be elegant code and simple to add. As it turned out, it was neither.

Lisp programmers may appreciate the inclusion of a List object, and the implementation of the Apply command, one of the nicer features of that wonderful language. Simply put, if you send a message to a List object it will dispatch the message to everyone in the list, getting a lot of work done with one call. The list structure is implemented with a dynamically resizing array, instead of the traditional doubly linked list. This makes it possible to include objects in any number of lists, with minimum overhead and maximum speed.

Fitting the definition

Now, let us re-examine the definition of Object Oriented Programming, and establish the extent to which POOPShell satisfies the textbook. I am not trying to establish the POOPShell as an example of true Object Oriented Programming, rather trying to teach something about Object Oriented Programming by showing what it is and what it isn't. I don't want to convince anyone to adopt my methods instead of C++, rather I want to ease the transition to that richer language through a half-way implementation. I do claim that if you shift from traditional program structure to a POOP-ier structure, the ultimate conversion to C++

might be easier.

Data Abstraction

The primary tenet of object oriented programming is Data Abstraction. This is a very fundamental concept, meant to avoid incorrect access of data. All C programmers employ it, but there are limitations to data abstraction in C which C++ will transcend.

The most significant "breakthrough experience" I had in my education into programming was when I made the transition from Basic to Pascal, and my introduction to structured programming. Here was a language and a vocabulary which embodied all of the good practices I struggled to maintain in my early programs. In many ways, Pascal enforced good programming style upon me, requiring me to declare variables, and making me be aware of which constructs were static (declared in the code) and which were dynamic (created with new statement and managed through pointers). As I became familiar with all the added rules and restrictions, I realized the extra work paid off immensely. The biggest impact was that I was no longer simply applying instructions to a machine; suddenly I was modelling a problem. The addition of complex data structures, made up of an infinitely expandable list of simpler ones united the process of problem solving with coding, and gave me a new insight into the interrelation between data structures and algorithms.

Now, I see that structured programming is only a midway station en route to object oriented programming. The elegance of structured programming still does not completely encapsulate data from accidental intrusion, it does not completely embody the correlation between the data and algorithms, even though this is an integral part of the conceptual thrust of Pascal (as evidenced by the title of Wirth's book: *Data Structures + Algorithms = Programs*).

We all are familiar with the difference between using local and global variables. Local variables are admittedly more trouble, as they require declaration in each routine (and much of compile-time debugging is devoted to declaring local variables), but the security and clarity they provide is almost universally accepted as worth the extra effort. What local variables do not provide, though, is the ability for two or more routines to access a variable, but still have it be "local" insofar as that other routines cannot change it. This is accomplished in true Object Oriented Language's by encapsulating the data within the object and only allowing access through a method associated with the object.

C does not offer quite that clear a definition, but, by limiting the scope of declarations to a single compile-unit or module, this is sufficiently accomplished. If you follow the common sense rule that an object's structure definition is declared in a single module and never referenced externally, then no one outside of the module can access those fields from outside the methods of the object. In this way, the C compiler will catch any accidental references to the private fields of an object. In a simple sense, the difference between data abstraction in C and C++ is not terribly significant. The latter allows more control over the level of protection, but the former offers plenty to those who are willing to obey the common sense rules of good programming style.

Inheritance

Whereas data abstraction is probably nothing new to you, it is quite likely that inheritance is a new concept. Inheritance is defined as the ability for an object to be specified as a descendent of another object, and, in the absence of overriding instructions, to assume the behaviour of the ancestor. As an example, all objects in POOPDraw have a bounds rectangle associated with them. Therefore, when I want to access the bounds rectangle of a text box, I do not have to write code which performs that function. The textbox, as a descendent of the standard object, will inherit that function, unless I specify that it should not.

If you tend to cut and paste code around to create new applications from old ones (and who doesn't), then it won't take you long to really benefit from the inheritance properties of C++. Once you establish an object in an old application, that piece of code, tested and debugged, will be usable in your newer applications. You call the older version the parent, and only the changes needed for newer application need to be written from scratch.

Inheritance is very simple to implement in this program. We have already discussed how all of the known messages understood by an object are implemented as case clauses within a switch statement. All inherited capabilities are accessed through the default clause. In traditional C programming, if none of the cases match, either nothing happens or, if the programmer is conscientious, an error handler is invoked. In this program, if the object does not understand a message, it will simply pass the message along to its parent class. At the topmost level, it is considered an error if a message is unrecognized, but at all lower levels, the default case is simply a call to another jump table.

Normally, the relationship of an object to a class is analagous to the that between an individual and her race. An object is an instantiation of a class. In true Object Oriented Languages, the parent of an object is an object itself, with its own data. In the POOPShell implementation, the parent is a class, but not an object. Instead, the class specifies where to look for a method if it is undefined, instead of naming an object which may know. This interferes with the implementation of dynamic binding, as discussed below, but is sufficient for providing the property of inheritance, and decreases the required message-passing overhead.

Seeing the tree structure of inheritance can be obscured by the separate tree structure associates with the dispatch of events down from the main event loop. Figure 1 shows a downward flow of execution first through "event links" and then through "inheritance links".

Polymorphism

Polymorphism is defined as the ability for different objects to respond differently to the same message. As an example, an Oval and a Line will not execute the same code in response to a command to DRAW, but they will both understand the command.

In C, any function declaration preceded by the key word static, specifies that the function may be called only within the module where it is declared. The linker will not complain if

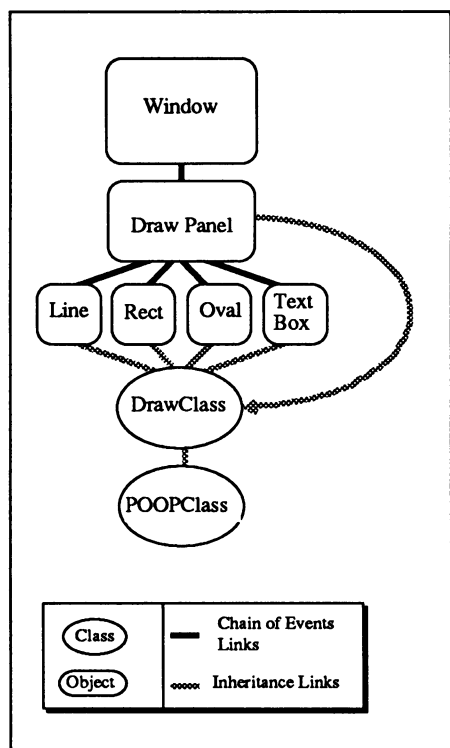


Figure 1

multiple static functions with the same name exist in different modules of a program. That is to say, C is a polymorphic language.

That is not to say that C++ isn't better. C++ incorporates *overloading*, which allows more than one non-static function to have the same name, and different parameters. I plan to use this feature extensively with Quickdraw, to reduce the common practice of breaking points into h and v, or transforming two points into a rectangle. But there is nothing that overloading can do that you can't manage with mangled pointers or #define. Therefore, it can be said that C programming in general, and the POOPShell in particular, are truly polymorphous perverse.

Dynamic Binding

The final criterion of true Object Oriented Programming, and the one omitted from this implementation, is dynamic binding. With dynamic binding, it is possible to change, at run-time, an object's inheritance path or its response to a message. In my mind, this is conceptually rather than technically difficult. An example of dynamic binding in practice is the way HyperCard attaches scripts to buttons and fields. In the course of running the program, the user is able to change the behavior of buttons and fields. But, apart from environments which are user programmable and hypertextual, there aren't many examples of dynamic binding obvious in programs I know of. So, brazen in my ignorance, I have chosen to wait to implement this feature until someone can give me a good reason to do so.

The method to implement dynamic binding is relatively simple. If we were to change the `dispatch` function pointer of an

object to that of a different object, from that point on, the object would behave differently. So, adding dynamic binding to POOPDraw is as simple as adding a method to change the dispatching function. (You should also change the class field, so that the object does not suffer an identity crisis, but that actually may not be necessary.)

Adding dynamic binding would require a only few substantial changes to the structure of POOPShell. In the current implementation, when the default clause of an object's dispatching function does not recognize a message, it calls a parent function, whose name is hard-coded into the `dispatch` function. To allow dynamic binding, the object would instead contain a field naming its parent. The parent itself would be a true object, requiring a globally declared object for each class to be created at start-up. Then, in the case a message is not recognized, the object would call:

```
Dispatch((ObjectH)->parent, Message, ParmP);
```

Dynamic binding would be achieved by giving the object the ability to change the value stored in the parent field.

One interesting advantage of this alternate method, would be that it would be possible to name not a single parent but multiple parents of an object. That is, our "language" would implement *multiple inheritance*, where an object can be defined as the descendent of several classes and have the abilities of all of them. As an example, a tear-off palette might be the descendent of a window, a menu, a picture and a rectangle. It would understand the messages to any of these elements without any code unique to itself. Multiple inheritance, a gem of conceptual gadgetry, employed by only the most object oriented purists, and lost on "the rest of us", is not implemented in the upcoming release of C++.

Objects in the Real World

This may be a good place to discuss the real world considerations of Object Oriented Programming. It is a fine line between how much is good modularity and the point when the concepts may instead burden the implementation. A good example is a rectangle. Here I don't mean the graphical element in POOPDraw, but rather the Quickdraw rectangle, as is in the bounds field in every object. In Smalltalk, the truest object oriented language I know of, every data structure, be it a window, a rectangle or an integer, is an object. This adds substantial overhead to every operation, and bogs down the system fairly quickly. As a C programmer, I want my code to be fast. Even when the difference is only a few instructions, and may be unnoticeable to the user, the code seems to read better (to me) when I know it doesn't unnecessarily waste time or memory.

If all of the rectangles in this program were objects instead of Rects, they would each require about four times the memory as a simple static structure. Because the rectangle is a parent to just about everyone, dispatching any message to a rectangle object would mean traversing several jump tables, dereferencing all kinds of handles, pushing several activation records onto the stack, and popping them all off on the way out. The time required

to perform:

```
Dispatch(rectangle,WIDTH,NULL,&answer);
```

could easily be a hundred times as long as

```
rectangle.right - rectangle.left.
```

I am willing to expend the extra overhead to have a function Width(r) in my library, but not to have a true object. This is a personal compromise, but it is my suspicion that the transition from C to C++, will be an exercise in balancing speed and familiarity vs. elegance and modularity. Calling statically bound methods in C++ is optimized and much faster than it would be in the POOPShell, but it still takes over twice as long as calling a function. My opinion is that, in the case of simple program elements (like numerical types, strings, rectangles, etc.), objects libraries should not replace standard function libraries, especially if you consider the fact that your function libraries are already written, debugged and understood. Remember, those are the attributes which attract us to C++ in the first place. It would be counter-productive to abandon methods that work and that are fast, for the sake of conceptual purity.

To Boldly Go...

The real beauty of C++ is that it is a superset of C, and any program you have should compile and run in C++ without modification. The pressure for change all at once is minimal. But the new language adds many of the extensions C has been missing, and opens the door to a new paradigm of programming. Look forward to a broadening mental transformation, as the C vocabulary expands onto new and vast conceptual frontiers.

[Due to the size of the application Adam submitted, we are unable to publish in the journal a full listing. Save your fingers and get the source code disk if you want POOPDraw. The following is a partial listing to give you some idea of the application. -ed]

Listing: Main.c

```
/******  
/* THE POOP SHELL */  
/******  
/* An Adventure in Pseudo-Object-Oriented-Programming */  
/******  
/*  
* >>> File name: Main  
* >>> Purpose: The main, start-up and shutdown routines  
* >>> Project: PoopDraw  
* >>> Date: June 5, 1989  
* >>> By: Adam Treister  
*/  
/******  
/*For Your Information 1802 Hillside Rd. SB CA 93101 */  
/******  
#include "PoopDrawInc"  
  
/******  
  
main (void);  
private void WakeUp (void);  
private void Die (void);  
  
foreign void Twiddle (void);  
foreign void ApplicationInit (void);
```

```
foreign void ApplicationShutDown (void);
```

```
/******  
/* THE GLOBAL DECLARATIONS */  
/******  
  
MenuHandle Menus[NUMMENUS];  
EventRecordEvent; /* the Event Record */  
long LastMouseDown = 0; /* time of last mouse down */  
Boolean MillerTime = false; /* Quit flag */  
int errno;  
  
/* ***** */  
/* MAIN */  
/* The standard shit - init, run, shutdown */  
/* ***** */  
main()  
{  
    WakeUp();  
    Twiddle();  
    Die();  
}  
  
/* ***** */  
/* WAKE UP */  
/* This function initialises the Mac Toolbox, creates */  
/* menu bar and calls application specific init routine */  
/* ***** */  
void WakeUp()  
{  
    WindowPtr win;  
    register int i;  
    extern MenuHandle Menus[];  
  
    InitMacintosh();  
    TurnWatchOn();  
  
    for (i=0; i < NUMMENUS; i++)  
    {  
        Menus[i] = GetMenu(i + AppleMenuID-1);  
        InsertMenu(Menus[i], 0);  
    }  
    AddResMenu(Menus[0], 'DRVR');  
    DrawMenuBar();  
    HiliteMenu(0);  
    ApplicationInit();  
    TurnArrowOn();  
}  
  
/* ***** */  
/* DIE */  
/* Tidys up everything before the application quits. */  
/* If there any windows open they are closed. */  
/* ***** */  
void Die()  
{  
    Str255 message,ctStr,sizStr;  
  
    while (MyFrontWindow())  
    {  
        WDispatch(MyFrontWindow(),CLOSE,NULL);  
    }  
    ApplicationShutDown();  
    ExitToShell();  
}  
  
/* ***** */  
/* send messages to the appropriate object */  
/* ***** */  
void Dispatch(ObjectH,Message,ParamP)  
{  
    ObjectHandle ObjectH;  
    int Message;  
    LPtr ParamP;  
    {  
        if (ObjectH)  
            (*(ObjectH)->dispatch)(ObjectH,Message,ParamP);
```

```

}
/*-----*/
/*A special case of Dispatch exclusively for windows.
/*-----*/

```

```

void WDispatch(wP,Message,ParamP)
WindowPtr wP;
int Message;
LPtr ParamP;
{
    ObjectHandle obj;
    obj = (ObjectHandle) GetWRefCon(wP);
    if (obj) (*(obj)->dispatch)(obj,Message,ParamP);
}

```

Listing: EvtHandlers.c

```

/*****
/* THE POOP SHELL */
/*****
/* An Adventure in Pseudo-Object-Oriented-Programming */
/*****
* >>> File name: EventHandlers
* >>> Purpose: main event loop (Twiddle) and other high
              level event handlers
* >>> Project: Briefcase
* >>> Date: April 5, 1989
* >>> By: Adam Treister
*/
/*****
/*For Your Information 1802 Hillside Rd. SB CA 93101 */
/*****
#include "PoopDrawInc"

void DoMenuCommand(long MenuSelectResult);

public void Twiddle (void);
private void DoMouseDown (EventRecord Event);
private void DoMouseUp (EventRecord Event);
private Boolean ClickIsInActiveWindow (WindowPtr ClickedWin-
dow);
private void DoKeyDown (EventRecord Event);
private void DoUpdate (WindowPtr wP);
private void DoActivate (EventRecord Event);
private void DoSuspendResume (void);

/*-----*/
/* MAIN EVENT LOOP */
/*-----*/

void Twiddle()
{
    extern EventRecord Event;
    extern Boolean MillerTime;
    register long Sleep = 10;
    register Boolean EventPending;
    register WindowPtr wP;

    while (!MillerTime)
    {
        EventPending = WaitNextEvent(everyEvent,
        &Event,Sleep,NULL);
        if (EventPending)
        {
            switch (Event.what)
            {
                case mouseDown: DoMouseDown(Event); break;
                case mouseUp: DoMouseUp(Event);
            }

            break;

            case keyDown:
            case autoKey: DoKeyDown(Event);
            break;

            case updateEvt: DoUpdate((WindowPtr)Event.message);
            break;

```

```

        case activateEvt: DoActivate(Event);
        break;
    }
}

/* if EventPending*/
else
{
    if (OurWindow(wP = MyFrontWindow()))
        WDispatch(wP, IDLE, NULL);
}

Boolean DoubleClicked; /* last mouse down was a double */
private long LastMouseUp;
private Point LastMouseUpLoc; /*not used for now */

/*-----*/
/* DO MOUSE DOWN */
/*This function handles mousedown event. FindWindow is */
/*called to find which window the mouse is in. Then event */
/*is "Dispatched" there. */
/*-----*/

void DoMouseDown(Event)
EventRecordEvent;
{
    WindowPtr WhichWindow; /* the event window */
    register Point MousePosition; /* current mouse pos */
    register int Where; /* the result of FindWindow */
    Rect DragRect;
    long okay = TRUE;

    DoubleClicked = (Event.when - LastMouseUp < GetDb1Time());

    MousePosition = Event.where;
    Where = FindWindow(MousePosition, &WhichWindow);

    switch (Where)
    {
        case inDesk: break;

        case inMenuBar:
            DoMenuCommand(MenuSelect(&Event.where));
            break;

        case inSysWindow: SystemClick(&Event, WhichWindow);
            break;

        case inContent: if (ClickIsInActiveWindow(WhichWindow))
            WDispatch(WhichWindow,MOUSEDOWN,NULL);
            break;

        case inDrag: DragRect = screenBits.bounds;
            DragWindow(WhichWindow,MousePosition, &DragRect);
            break;

        case inGrow: WDispatch(WhichWindow,GROW,NULL);
            break;

        case inGoAway: if (TrackGoAway(WhichWindow, MousePosi-
            tion))
            if (_OptionKeyDown(Event))
                while ((WhichWindow = MyFrontWindow()) AND okay)
                { WDispatch(WhichWindow,CLOSE,&okay);}
            else WDispatch(WhichWindow,CLOSE,&okay);
            break;

        default:
            Oops("\pUnknown Window Type in MouseDown Handler", 0,
            TRUE);
            break;
    } /* switch (FindWindow..) */
}

/*-----*/
/* DoMouseUp */
/*Set current time into LastMouseUp for Double Click */
/*Detection in DoMouseDown. */
/*-----*/

```

```

void DoMouseUp(Event)
EventRecordEvent;
{
    extern long LastMouseUp;
    LastMouseUp = Event.when;
}

/* ----- */
/* Boolean ClickIsActiveWindow(ClickedWindow)*/
/*A quickie to select back windows if the're clicked. */
/* ----- */
#define ItIsnt (!ItIs)

Boolean ClickIsActiveWindow(ClickedWindow)
WindowPtr ClickedWindow;
{
    register Boolean    ItIs;
    long type;

    ItIs = (ClickedWindow == FrontWindow());
    if (ItIsnt) SelectWindow(ClickedWindow);
    return(ItIs);
}

/* ----- */
/*      DO KEY DOWN      */
/*This function handles a keydown event.  If command */
/*key is down its a menu command, otherwise dispatch it */
/* ----- */

void DoKeyDown(Event)
EventRecordEvent;
{
    register char      key;

    key = Event.message & charCodeMask;

    if (_CmdKeyDown(Event))    DoMenuCommand(MenuKey(key));
    else if (MyFrontWindow())
WDispatch(MyFrontWindow(),KEYDOWN,NULL);
}

/* ----- */
/*      DO UPDATE      */
/* ----- */
void DoUpdate(WP)
register WindowPtr  WP;
{
    GrafPtr      PortSave;

    GetPort(&PortSave);
    SetPort(WP);
    BeginUpdate(WP);
    WDispatch(WP,UPDATE,NULL);
    EndUpdate(WP);
    SetPort(PortSave);
}

/* ----- */
/*      DO ACTIVATE      */
/* ----- */
void DoActivate(Event)
EventRecordEvent;
{
    register WindowPtr  ActivateWindow;
    register int        Message = DEACTIVATE;

    TurnArrowOn();
    ActivateWindow = (WindowPtr) Event.message;

    if (Event.modifiers & activeFlag)
        /* Activate vs Deactivate */
    {
        SetPort(ActivateWindow);
        Message = ACTIVATE;
    }
    WDispatch(ActivateWindow,Message,NULL);
}

```

```

}

```

Listing: WIND Draw.c

```

/***** SOURCE CODE FILE *****/
/*****
* >>> File name: 5.1 DrawWindow.c
* >>> Purpose: Methods for Rectangle Object
* >>> Project: PoopDraw Version 1
* >>> Date: 2/20/89
* >>> By: Adam Treister
*/
/*****For Your Information 1802 Hillside Rd. SB CA 93101 *****/
/*****
#include "PoopDrawInc"

void DrawOutline(Rect r,int curTool);

typedef struct
{
    _StdObjectFields
    ObjectHandle drawPanel;
    ControlHandle vScrollBar,hScrollBar;
    Point curOrigin;
    ObjectHandle doc; /* info about files and printing */
} WindowDataRec,*WindowDataPtr,**WindowDataHandle;

/***** Public Functions *****/
/* WindowPtr NewDrawWindow(void); */
/***** Private Functions *****/

DrawWinDispatch(WindowDataHandle ObjectH,int message,LPtr
ParmP);
WindowPtr NewDrawWindow(void);
private void Dispose(WindowPtr WP);
private void MouseDown(WindowPtr WP);
private void Grow(WindowPtr WP);
private void KeyDown(WindowPtr WP);
private void Update(WindowPtr WP);
private void Activate(WindowPtr WP);
private void DeActivate(WindowPtr WP);
private WindowDataHandle GetWinData(WindowPtr WP);

/***** Local Defines & Includes *****/
#define UntitledWindowName"\pUntitled"

/*****
DrawWinDispatch(ObjectH,message,ParmP)
WindowDataHandle ObjectH;
int message;
LPtr ParmP;
{
    WindowPtr WP;
    WP = (*ObjectH)->port;
    switch (message)
    {
        case CLOSE:
        case DISPOSE: Dispose(WP); break;
        case MOUSEDOWN:MouseDown(WP); break;
        case UPDATE: Update(WP); break;
        case ACTIVATE: Activate(WP); break;
        case DEACTIVATE: DeActivate(WP); break;
        case GROW: Grow(WP); break;
        default: Dispatch((*ObjectH)-
>drawPanel,message,ParmP);
    }
}

/* ----- */
/*      New Draw Window      */
/* ----- */

WindowPtr NewDrawWindow()

```

```

{
    WindowPtr    wP;
    Rect         BoundsRect;
    WindowDataHandle WinData;
    extern Boolean DEBUG;

    BoundsRect = screenBits.bounds;
    BoundsRect.top += 40;

    WinData = _GetHandleToRecord(WindowDataRec);
    NullOutHandle(WinData);
    wP = NewWindow (NULL, &BoundsRect, UntitledWindowName,
        true, documentProc, -1L, true, (long) WinData);

    SetPort(wP);
    (*WinData)->port = wP;
    (*WinData)->dispatch = DrawWinDispatch;

    New(DRAWPANEL, WinData, &(*WinData)->drawPanel);
    return (wP);
}

/* ----- */
/* Dispose Document Window */
/* ----- */
void Dispose(wP)
WindowPtr wP;
{
    WindowDataHandle TheWindowData = GetWinData(wP);

    Dispatch((*TheWindowData)->drawPanel, DISPOSE, NULL);
    DisposeHandle(TheWindowData);
    DisposeWindow(wP);
}

/* ----- */
void MouseDown(wP)
WindowPtr wP;
{
    short    PartCode;
    ControlHandle ctrlH;
    extern EventRecord Event;
    register Point pt;

    pt = Event.where;
    GlobalToLocal(&pt);
    PartCode = FindControl(pt, wP, &ctrlH);
    /*if (PartCode)
        HandleScrollBars (wP, ctrlH, pt, PartCode);
    else
    */{
        WindowDataHandle WinData = GetWinData(wP);
        Dispatch((*WinData)->drawPanel, MOUSEDOWN, NULL);
    }
}

/*-----*/
void Grow(wP)
WindowPtr wP;
{
    WindowDataHandle WinData;
    long    newSize;
    Rect    oldRect, growZone;
    extern EventRecord Event;

```

```

    int    newH, newV;

    WinData = GetWinData(wP);
    growZone = screenBits.bounds;
    growZone.left = 100; growZone.top = 260;
    newSize = GrowWindow(wP, Event.where, &growZone);
    if (!newSize) return;

    newH = LoWord(newSize);    newV = HiWord(newSize);
    SizeWindow(wP, newH, newV, TRUE);
    Dispatch((*WinData)->drawPanel, RESIZE, NULL);
    ClipRect(&wP->portRect);
    EraseRect(&wP->portRect);
    InvalRect(&wP->portRect);
}

/* ----- */
/* Update */
/* ----- */
void Update(wP)
WindowPtr wP;
{
    WindowDataHandle WinData = GetWinData(wP);
    Rect r;

    r = wP->portRect;
    ClipRect(&r);
    DrawControls(wP);    /* there are none, unless you add
        scrolling */
    DrawGrowIcon(wP);
    r.right -= ScrollBarWidth; r.bottom -= ScrollBarWidth;
    ClipRect(&r);
    Dispatch((*WinData)->drawPanel, UPDATE, NULL);
}

/* ----- */
/* Activate */
/* ----- */
void Activate(wP)
WindowPtr wP;
{
    /* activation of controls goes here */
}

/* ----- */
/* Deactivate */
/* ----- */
void DeActivate(wP)
WindowPtr wP;
{
    /* deactivation of controls goes here */
}

/* ----- */
/* GetWindowDataHandle */
/* This get WindowDataHandle from the refCon field of */
/* the window passed to the function. */
/* ----- */
WindowDataHandle GetWinData(wP)
WindowPtr wP;
{
    return((WindowDataHandle) GetWRefCon(wP));
}

```



Jörg's Folder

C++ Overview



Jörg Langowski
MacTutor Editorial Staff
Volume 5, Number 9

"An Overview Of C++"

MacHack '89 brought me not only a colorful set of screwdrivers from APDA, but also a new assignment: yours truly is supposed to run a tutorial column on C++. C++ is a very interesting programming language. Supposedly, the new Finder was written in it. Also, it exists on a couple of Unix systems. In fact, shopping for a Unix system, we recently met a representative who assured us that C++ would be delivered with the system. Jordan Matthews, and other people from Apple, spoke very highly about C++ at the MacHack, and assured us that we would get our fingers on a pre-release of Apple's C++ for MPW, supposedly to be delivered by the end of this year.

As you might have guessed, Apple hasn't sent us the pre-release yet, and I have yet to use a working C++ compiler. So far my only 'hands-on' experience is Bjarne Stroustrup's book, *The C++ Programming Language* (Addison-Wesley 1987), which I highly recommend.

The style of the book is rather terse, and you have to work your way through. A good example is that after the introduction, not much is said about 'object-oriented' programming, until you hit page 213:

"A list specified in terms of pointers to a class can hold objects of any class derived from that class. That is, it may be heterogeneous. This is probably the single most important and useful aspect of derived classes, and it is essential in the style of programming presented in the following example. That style of programming is often called object based or object oriented; it relies on operations applied in a uniform manner to objects on heterogeneous lists."

So now you know what you've really done when you used MacApp... The fact that Stroustrup refers to object-oriented programming in this rather abstract way made me dig out an old introduction to Simula 67, which was the first language to introduce object-oriented concepts. There, too (the book dates from 1973) no reference is made to OOP as we know it today. All the important constructs - classes, instances, methods, overriding - are already there, and one could have implemented today's programming style in Simula; only computers were much smaller, and most programs did not demand OOP concepts.

The C++ Design

Stroustrup's team designed C++ for dealing with simulation problems not unlike those that Simula was developed for. However, C++ is a much broader concept than simply a set of 'object-oriented' extensions to C; it is a redesign of the C language. To use another quotation from Stroustrup's book, *"C++ was designed to enable larger programs to be structured in a rational way so that it would not be unreasonable for a single*

person to deal with 25,000 lines of code". To achieve such an ambitious goal, the most important point is to allow the user to extend the language to accommodate new 'shorthand notations' for things that have to be done over and over again. For instance, in a program that uses matrix algebra, given the 25-row by 35-column matrix C and 25-row by 15-column matrix B,

$$A = B * C$$

is much easier to read than

matmul (A,transpose(B),C,15,25,35).

To be able to use such a shorthand for matrix multiplication, we would need two features built into the language: a, data structures that carry additional information, such as row and column size for a matrix, but which is normally hidden to the user; and b, the capability to redefine operators - like '*' or '^' - depending on the context in which they are used. The latter feature would then cause a '*' to behave differently depending on whether it is used to multiply two integers, reals, vectors or matrices. Some of this behavior is already built into most compilers: integer and real multiply generate different code. But this behavior cannot be modified. C++ allows you to modify your operators in any odd way.

Classes

Let's assume we wanted to define an array structure, matrix, whose size is not defined at compile time and for which space will be dynamically allocated at run time. In C, one might write

```
typedef struct matrix
{
    int rows, cols;
    int *m; /* pointer to matrix data */
}
```

matrix a
and then write an indexing function elem(i,j) which refers to the (i,j)th element of the matrix a by

```
*(a.m + 2*(i*a.cols + j)).
```

Of course, it would be much easier to simply define a two-dimensional array and write a[i][j], but let's stay with this definition for a while; unlike the usual array definition, this matrix is resizeable and space is allocated dynamically at run time. We would have to find a block of memory to hold the matrix data and put a pointer to it in m.

When we access an array, we are often not interested in its actual dimensions, as long as the indices are not out of range. In C++ we can define the matrix type in such a way that only certain functions have access to information 'private' to the array (such as its dimensions), and all access to the array's data is done

through these access functions, called *methods*. Data structures that may carry private information are called *classes* in C++. (According to the manual, classes are 'user-defined types' - the most general definition that one might imagine!). A class is just like a *struct* in which some of the fields cannot be 'seen from the outside', and in which the interface to these private fields is defined through method declarations. The C++ class definition for the matrix type would look very similar to a struct declaration, with some additions. The syntax of the class declaration is:

```
class matrix
{
    int rows, cols;
    int *m; /* pointer to matrix data */
public:
    int rowsize() { return rows; }
    int colsize() { return cols; }
    void set_size(int, int);
    int& elem(int, int);
    matrix(int, int);
    ~matrix;
}
```

matrix a

Those of you who have had some experience with NEON [let's make the point again that it is a shame that NEON has disappeared...] might recall that its class definition looked similar:

```
:class matrix <super object
    2 <indexed
    int rows
    int cols
    :M rowsize ..... ;M
.....
etc...
```

:class

NEON, however, did not have the label public: for separating the private and public parts of the class declaration. In NEON, all variables were private and all the methods were public.

The C++ class declaration is similar to a C struct declaration, with the possibility to include functions and to hide parts of the declaration from the outside. The public functions in a class that constitute the interface to the outside world are called *methods*.

There are two principal ways to define a method. One can write the method code inside the class declaration (as for rowsize and colsize in the example above), or one can just declare the method and write the method code later, as for set_size or elem. elem returns the reference to an integer that is the (i,j)th element of the matrix and might be defined as follows:

```
int& matrix::elem(int i, int j) { return m[i*cols + j]; }
```

There is a fundamental difference between methods defined inside and outside of a class declaration. The methods defined outside will be called through a subroutine call, while inside-defined methods are inline-expanded by the compiler. Writing a.rowsize will not generate a JSR to the function code, but code that will directly reference the hidden field a.row. However, any method that is defined outside a class declaration may also be defined as an inline method by prefixing it with the keyword inline:

```
inline int& matrix::elem(int i, int j) { return m[i*cols + j]; }
```

There are two more special methods in the class declaration which carry the name of the class, or respectively the class name prefixed with a tilde (~). These are the so-called constructor and destructor methods; they are called when a new object is declared (as in matrix a;) or deleted (when one leaves the block that the object was declared in). Constructors and destructors are important when heap space has to be allocated for an object (our matrix will need it) and deallocated when the object is no longer defined.

Operators

Our dynamically sized matrix might be defined in a slightly different way which allows to access the elements in the usual way, writing a[i][j] instead of a.elem(i,j). One first defines a one-dimensional array class (as in Stroustrup's book):

```
class vector
{
    int* v;
    int sz;
public:
    vector(int); ~vector();
    int size () { return sz; }
    void set_size(int);
    int& operator[] (int);
    int& elem(int i) { return v[i]; }
};
```

and then builds the two-dimensional class on top of it:

```
class matrix : vector
{
    vector*& mv;
    int rows, cols;
public:
    matrix(int, int); ~matrix;
    int rowsize () { return rows; }
    int colsize () { return cols; }
    void set_size(int, int);
    vector*& operator[] (int);
    int& elem(int i, int j) { return mv[i][j]; }
};
```

(I hope this is approximately correct while I'm waiting for the C++ system to try this out and get ready for your embarrassing remarks). In the program, one would declare matrix a(10,20) and access the (i,j)th element by writing a[i][j]. The array indexing operator [] has been re-declared in the class declaration, and will now support checking of index bounds, if we wish so.

The actual implementation of the operators has of course to be done separately. We would write

```
int& vector::operator[] (int i) { /* body of code */ }
and
vector*& matrix::operator[] (int i) { /* body of code */ }
to implement the new definitions.
```

The matrix multiplication operator may now be defined easily. We write

```
matrix operator*(matrix& a, matrix& b)
{
    matrix c(a.colsize,b.rowsize);
    if (a.rowsize != b.colsize) error "index mismatch";
    for (int i=1; i<a.colsize; i++)
        for (int j=1; j<b.rowsize; j++)
            {
```



```

        int sum = 0;
        for (int k=1 ; k<a.rowsize ; k++)
            sum = sum + a[i][k]*b[k][j];
        c[i][j] = sum;
    };
    return c;
};

```

Again, I hope this would work in an actual example. It is not the most efficient way to program the matrix multiplication; the good way to do it would be using friend definitions. This concept is explained in Stroustrup's book, and I'm going to come back to it in the next column, where I can supply some examples.

The expression $a*b$, where a and b are of type matrix, would return a pointer to another object of class matrix, which contains the product of a and b . To make sense of the expression $c = ^a*b$, we would also have to define the transpose operator, '^', and the assignment operator, '='. I won't write these definitions down here; you might try to work them out, or better, test them if you

have a C++ system available.

Operator redefinition is one of the most important concepts of C++, since it makes the code much more readable. The redefinition of an existing operator (like +, *, etc.) is called operator *overloading*; when such a redefined operator is used, the compiler will automatically search the existing definitions to find one that works on the data types provided. Thus, even if one redefined * for matrices, integer and real multiplications would still work as before. I have not found out yet whether dynamic binding is possible for operators by declaring them virtual (see below), but I'm sure I'll soon be able to test that.

Class Hierarchies

We have seen the syntax of a class definition which was derived from another class, class matrix : vector { }. If we define a derived class this way, none of the methods in the superclass

A sample class definition

Object Pascal

```

type
  TShape =
    Object (TObject)
      Next: TShape;
      Prev: TShape;
      boundRect: Rect;
      ShapeRgn: RgnHandle;
      procedure Create(theRect: Rect);
      procedure Track(oldRect, newRect: Rect);
      procedure Draw;
      procedure Erase;
      procedure Free; Override;
    end;

  TCircle =
    Object (TShape)
      procedure Create(theRect: Rect); Override;
      procedure Track(oldRect, newRect: Rect);
        Override;
      procedure Draw; Override;
    end;

  TShapeWindow =
    Object (TObject)
      HeadofList: TShape;
      TailofList: TShape;
      procedure Init;
      function NewShape(knd: integer): TShape;
      procedure Add(Shape: TShape;
        theRect: Rect);
      procedure SelectAndMove(thePoint: Point);
      procedure Delete(thePoint: Point);
      procedure Update;
      procedure Free; Override;
    end;

```

C++

```

class TShape
{
    TShape* Next, Prev;
    Rect boundRect;
    RgnHandle ShapeRgn;

public:
    virtual void Create(rect *theRect);
    virtual void Track(rect *oldRect,*newRect);
    virtual void Draw();
    virtual void Erase();
    virtual void Free();
}

class TCircle : public TShape
{
public:
    virtual void Create(rect *theRect);
    virtual void Track(rect *oldRect,*newRect);
    virtual void Draw();
}

class TShapeWindow
{
    TShape* HeadofList,TailofList;

public:
    virtual void Init();
    virtual TShape NewShape(int knd);
    virtual void Add(TShape *Shape; rect *TheRect);
    virtual void SelectAndMove(Point thePoint);
    virtual void Delete(Point thePoint);
    virtual void Update;
    virtual void Free;
}

```

will be accessible through an object of the subclass; all subclass methods have to be explicitly defined in the subclass declaration. If we write, on the other hand, `class matrix : public vector { }`, any method from class `vector` that is not redefined in class `matrix` is usable on objects of class `matrix` as well. This is the way we very often wish objects to behave; methods that are redefined in a subclass should override the superclass definition, but if an object does not 'know' about a method it should look for a definition higher up in the hierarchy.

In a class hierarchy we should therefore be able to apply a method to an arbitrary object whose exact type is not known at compile time. If the object's type is known at compile time, the compiler will simply generate a JSR to the appropriate method code, passing arguments as required. This is known as *early binding* in object-oriented jargon. If the type is not known, we must check at run time what type of object is given the method call, and see whether the method is defined in the object's class declaration or somewhere higher up in the hierarchy. This is called *late binding*; a run time error message will be generated if the method can't be found for a particular object.

Late binding is important if we have a list of objects to which the same method should be applied, for instance a list of shapes - rectangles, circles, polygons - to be drawn on a screen. If the list is kept in an array `shapelist[i]`, we could then simply write

```
for (i=1; i<=N; i++) shapelist[i].draw;
to draw all the objects. This is very similar to Object Pascal,
```

where we would write analogously

```
for i :=1 to N do shapelist[i].draw;
```

However, in Object Pascal late binding is always used when early binding can't be applied. In C++, we have to tell the compiler that a method could be used for late binding by declaring it `virtual`:

```
class TShape {
    TShape* Next, Prev;
    Rect boundRect;
    RgnHandle ShapeRgn;
public:
    virtual void Create(rect *theRect);
    virtual void Track(rect *oldRect, *newRect);
    virtual void Draw();
    virtual void Erase();
    virtual void Free();
}
```

This is the generic definition of a shape for which methods for drawing, erasing, etc. exist, but may or may not be defined in the top class; they may be overridden in the descendant classes, and the actual binding may be known only at run time. The figure illustrates the definition of a class hierarchy of shapes in C++ and in Object Pascal.

This more or less concludes my quick overview of the main characteristics of C++ (of course, all the features of C are still present in the language). Don't laugh at the mistakes that are probably still in the examples; this happens when one writes programs without a compiler. There are many details I haven't gone into here; we'll get to know them in the following columns, with corresponding examples. Forth friends, don't despair; you'll get your share soon again, too.



C Workshop

Three Doodats



Lee A. Neuse
Manassas, VA
Volume 5, Number 9

[Lee Neuse started in the days you built them before programmed them. He was lured away from IBM compatibles to the Mac in 1982 and been there ever since. Currently, he is working for Computer Science Corporation designing and implementing Mac software as part of CSC research. His Mousehole handle is "Noisy".]

Three Doodats

Doodat—a trivial problem guaranteed to make a programmer's life miserable; from the expression "How did he do dat?"

It seems like every programmer, at one time or another, is faced with the same dilemma: wanting to use a feature in a program, but not wanting to spend the time to figure out the technique. For example, we all know that a Balanced B-Tree index is wonderfully efficient, but how many of us really want to sit down and write the code? One solution is to start reading back issues of MacTutor, hoping that some kind soul has already published useable source code. Another is to scour public domain software for an author willing to impart knowledge in exchange for a shareware donation. Either way, it's usually faster (and easier) to find and adapt someone else's code instead of doing it yourself.

The purpose of 'Doodats' is to provide generic solutions to the little problems, so the programmer can spend his or her time working on the big problems (like what the program does).

Doodat #1

Problem: how to determine if the user has tried to abort something by pressing the Cmd-'. ' keys. EventAvail() returns only the first keystroke event in the queue, and the user may have pressed some other keys since the last keystroke event was read by the application. Either GetNextEvent() or WaitNextEvent() return *all* keystroke events, forcing the application to store them or ignore them (a Bad Thing). Moreover, if MultiFinder is running, calling any of these event-related traps may cause the application to be switched out, which could cause problems.

Solution: Doodat #1 is a routine called check_abort(), which directly scans the ToolBox's internal event queue for Cmd-'. ' abort events.

The routine starts by setting a pointer to head of the event queue (stored in the system global EventQueue), then examining the next event in the queue. If the event is a Cmd-'. ' event, it is removed by calling Dequeue(), and the 'found' flag set. If the pointer does not match the event queue tail, it is advanced to the next event, otherwise, the function returns. This technique solves all of the problems posed by normal event processing:

1. It doesn't affect any other keystroke events in the queue.

2. If the user has generated multiple aborts (by holding the keys down), it removes all of them.

3. It does not call EventAvail() or WaitNextEvent(), so no chance of being switched in or out under MultiFinder.

To use check_abort(), call it from within any place where the user might want to abort. For example:

```
for (page = 1; page < last_page; page++)
{
    /* print a page */
    if (check_abort())
        break;
}

/*****
**
**check_abort()- Lightspeed C Version
**
**This routine returns scans the low-level
**event queue in search of a Cmd-'. ' key event
**Each one found is removed from the event
**queue.
**
**In: N/A
**
**Out: TRUE if Cmd-'. ' found, FALSE otherwise.
**
**/

Boolean
check_abort()
{
    EvQElPtr eq_p;
    Boolean f_found = false;

    /* start at head of internal queue */
    eq_p = (EvQElPtr)(EventQueue.qHead);

    while (true)
    {
        if ((eq_p->evtQWhat == keyDown ||
            (eq_p->evtQWhat == autoKey) &&
            (eq_p->evtQModifiers & cmdKey) &&
            (eq_p->evtQMessage & charCodeMask) == '. '))
        {
            /* remove the event from the queue */
            Dequeue((QElmPtr)eq_p, &EventQueue);
            f_found = true;
        }

        /* test for end of queue */
        if (eq_p == (EvQElPtr)EventQueue.qTail)
            break;

        /* continue with next queue entry */
        eq_p = (EvQEl *) (eq_p->qLink);
    }

    return(f_found);
} /* end of check_abort() */
```

Doodat #2

Problem: Many windows contain items such as lines, icons,

pictures, or text strings that are there for decoration; these items don't change in appearance, and clicking on them has no effect. While the code to draw these items is relatively simple, it is also very boring to write, and frequently quite lengthy. In addition, trying to figure out exactly which horizontal and vertical co-ordinates to use can be time-consuming and frustrating.

Solution: Doodat #2 is an attempt to save time (and code!) by making ResEdit and the Resource Manager do most of the work. This involves a two-step approach:

Step 1 is to use ResEdit (or any other Resource tool) to create a 'DITL' resource (Dialog Item List) that contains all of the items desired. ResEdit is particularly handy, as the items can be arranged visually within the window. The DITL resource may be created and edited normally, with all items being set to *disabled*. No Control or EditText items should be included (they will be ignored if you do), and the items may be in any order. The finished DITL resource is then included in the application's resources for later use.

UserItems in the item list are handled in one of two ways: either as a dividing line or as frame. If the horizontal or vertical co-ordinates are equal, i.e. the boundary rectangle is empty, it is drawn as a 50% gray line. If the rectangle is not empty, it is drawn as a black frame.

Text items are drawn (left-justified) in whatever font the window's port is using when draw_ditl() is called. Different text characteristics could be supplied by reserving the first few characters of the text string...but that's for another time.

Step 2 is to use the routine below called draw_ditl() to actually read in and draw the items within the window. This routine accepts the ID number of a DITL resource, reads the resource into memory, then traverses the item list, drawing each item accordingly. For draw_ditl() to work properly, the DITL resource must be present in the application's resource fork (or in an open resource file), and the port set to the desired window.

draw_ditl() uses the following definitions and structure:

```
#define NIL      0L

/* Test a pointer for validity */
#define VALID_PTR(p) ((long)(p) &&
    ((long)(p) & 1L) == NIL)

/* Test a handle for validity */
#define VALID_HNDL(h) (VALID_PTR(h) &&
    VALID_PTR((long)(h)) & 0x0FFFFFFF)

typedef struct
{
    Handle      hndl;
    Rect        frame;
    unsigned char  type;
    unsigned char  length;
    short        data[];
} ditl_list;

/*****
**
**draw_ditl()- Lightspeed C Version
**
**This routine reads in and draws the items in a
```

```
**Dialog Item List (DITL) resource.
**
**In: resource id of DITL
**
**Out: N/A
**
*/

void
draw_ditl(ditl_id)
short    ditl_id;
{
    Handle    ditl_h;
    ditl_list *ditl_p;
    PenState  save_pen;
    short     d_type, offset, num_items;

    ditl_h = GetResource('DITL', ditl_id);
    if (ResError() != noErr || !VALID_HNDL(ditl_h))
        return;

    HLock(ditl_h);
    /* don't change pen behind the window's back */
    GetPenState(&save_pen);
    PenNormal();
    /* get the number of items in list */
    num_items = *((short *)(&ditl_h)) + 1;
    /* set pointer to first item in list */
    ditl_p = (ditl_list *)(&ditl_h) + sizeof(short);
    while (num_items--)
    {
        /* mask out the item disabled bit */
        switch (ditl_p->type & 0x7F)
        {
            case statText:
                TextBox((char *)ditl_p->data,
                    ditl_p->length, &(ditl_p->frame),
                    teJustLeft);
                break;

            case iconItem:
                ditl_p->hndl =
                    GetIcon(ditl_p->data[0]);
                if (VALID_HNDL(ditl_p->hndl))
                {
                    PlotIcon(&(ditl_p->frame),
                        ditl_p->hndl);
                    ReleaseResource(ditl_p->hndl);
                }
                break;

            case picItem:
                ditl_p->hndl = GetResource('PICT',
                    ditl_p->data[0]);
                if (VALID_HNDL(ditl_p->hndl))
                {
                    DrawPicture((PicHandle)ditl_p->hndl,
                        &(ditl_p->frame));
                    ReleaseResource(ditl_p->hndl);
                }
                break;

            case userItem:
                if (EmptyRect(&(ditl_p->frame)))
                {
                    PenPat(gray);
                    MoveTo(ditl_p->frame.left,
                        ditl_p->frame.top);
                    LineTo(ditl_p->frame.right,
                        ditl_p->frame.bottom);
                    PenPat(black);
                }
                else
                    FrameRect(&(ditl_p->frame));
```

```

    break;
}
/* force offset to a word (even) boundary */
if ((offset = ditl_p->length) & 1)
    offset++;
/* advance pointer to next item */
ditl_p = (ditl_list *)((char *)ditl_p +
    sizeof(ditl_list) + offset);
}

HUnlock(ditl_h);
ReleaseResource(ditl_h);
/* restore pen to original condition */
SetPenState(&save_pen);
} /* end of draw_ditl() */

```

Doodat #3

Problem: Sooner or later, an application is going to need to access a data file of some sort, with fixed-length or variable-length records, keys, and so forth. Using the Resource Manager as a Poor Man's Database is a Bad Thing according to Apple, and not everyone can afford (or write) a full ISAM (Indexed Sequential Access Method) utility.

Solution: Doodat #3, which includes some primitive (but useful) indexing routines. If used properly, they can be used for indexing files, arrays, or be used for look-up tables, etc.

All of the routines are based on the idea of a simple index, made up of 1-*n* index entries. Each index entry is a key value and it's associated data value, and is defined in the index_entry structure below. The key and data values are always handled in pairs: for any given key, there is only one data value (provided it exists). Duplicate keys are not allowed, and both key and data values are limited to numeric values. What the key and data values actually represent depends on the application, an obvious example would be relating a part number to an offset into a file.

Several different indices can be manipulated using the same routines, it is the application's responsibility to determine which index to use. If the indexes are small and/or few in number, they could be manually rebuilt each time the application is executed. Alternately, they could be saved between sessions, either as data files, or as resources: all that is needed the memory block referenced by the index handle (use GetHandleSize() to find it's size). An elegant solution would be to save any or all data file indices as resources in the data file's resource fork, but that would limit the index size to 32K.

Adding a new key to an index (or updating an existing key) is done by the db_insert_key() function: a pointer to the index handle, a key value, and the data value associated with the key are all passed as parameters. If the index handle is NIL, a new handle is allocated, otherwise, the index is expanded by one index entry, and the new key added to the index in ascending order. If the key value already exists in the index, the data value is updated with the new value.

Once an index has keys inserted into it, db_get_key() can be used to search the index for a given key value. The routine expects an index handle and a key value, then performs a binary search on the index. If the key value is found, the data value associated with the key is returned, otherwise -1 is returned to

signal failure.

There may be times when it is necessary to find a key's location in the index, thus db_get_seq_key() accepts an index handle and an entry number, then returns the key value found at that entry. Once again, -1 is returned to signal failure if the entry number is less than 1 or greater than the number of keys. This routine can be used to sequentially read through an index (in ascending order), using the following code:

```

long w_key, key_value;

w_key = 1L;
while (true)
{
    key_value = db_get_seq_key(my_index, w_key++);
    if (key_value == -1L)
        break;

    my_data = db_get_key(my_index, key_value);
    /*
    **process my_data here
    */
}

```

Some possible extensions to these routines include modifying them to store a character pointer as the key, providing non-numeric keys. For small indexes, the entire data record could be embedded as part of the index_entry structure, and a pointer to the structure returned by db_get_key().

The routines db_get_key(), db_get_seq_key(), and db_insert_key() all utilize the following definitions and structure:

```

#define NIL      0L
/* Test a pointer for validity */
#define VALID_PTR(p) ((long)(p) &&
    ((long)(p) & 1L) == NIL)

/* Test a handle for validity */
#define VALID_HNDL(h) (VALID_PTR(h) &&
    VALID_PTR((long)(h)) & 0x00FFFFFF)

typedef struct
{
    unsigned long key;
    unsigned long data;
} index_entry;

/*****
**
**db_get_key()
**
**This routine performs a binary search on a
**list of index entries
**
**In: index_hndl Handle to index
**     target    key value to find
**
**Out: -1 if NOT found, data value otherwise
**
**/

long
db_get_key(index_hndl, target)
Handle    index_hndl;
register long target;
{

```

```

register index_entry*i_ptr;
register long mid, low, high;

if (VALID_HNDL(index_hndl))
{
    low = 1L;
    high = GetHandleSize(index_hndl) /
        sizeof(index_entry);

    while (low <= high)
    {
        /* calculate the mid point of list */
        mid = ((low + high) >> 1) - 1;
        i_ptr = *index_hndl + (mid *
            sizeof(index_entry));

        if (i_ptr->key > target)
        /* too big, so look in lower half */
            high = mid;
        else if (i_ptr->key < target)
        /* too small, so look in upper half */
            low = mid + 2;
        else
        /* hmmm, guess we found it... */
            return(i_ptr->data);
    }
}
return(-1L);
} /* end of db_get_key() */

/*****
**
**db_get_seq_key()
**
**This routine reads a list of index entries,
**returning a key based on a sequential number
**
**In: index_hndl Handle to index
**     key_num   key number to return
**
**Out: -1 if NOT found, key value otherwise
**
**/

long db_get_seq_key(index_hndl, key_num)
Handle index_hndl;
short key_num;
{
    index_entry*i_ptr;
    long result = -1L;
    short num_keys;

    if (VALID_HNDL(index_hndl))
    {
        /* count number of keys in index */
        num_keys = GetHandleSize(index_hndl) /
            sizeof(index_entry);
        if (-key_num < num_keys)
        {
            /* calculate offset into the index */
            i_ptr = *index_hndl + (key_num *
                sizeof(index_entry));
            result = i_ptr->key;
        }
    }

    return(result);
} /* end of db_get_seq_key() */

/*****
**
**db_insert_key()
**
**This routine inserts a key into an index.

```

```

**
**In: index_hndl Handle to index
**     target    key value to insert
**     data      data value associated
**              with the key
**
**Out: N/A
**/

void
db_insert_key(index_hndl, target, data)
Handle *index_hndl;
long target;
long data;
{
    register index_entry*i_ptr, *n_ptr;
    register long mid, low, high, i_size;

    /* empty list is special case! */
    if (!VALID_HNDL(*index_hndl))
    {
        /* allocate new handle */
        *index_hndl = NewHandle(sizeof(index_entry));
        ((index_entry *)(*index_hndl))->key = target;
        ((index_entry *)(*index_hndl))->data = data;
    }
    else
    {
        low = 1L;
        high = i_size = GetHandleSize(*index_hndl) /
            sizeof(index_entry);

        while (low <= high)
        {
            /* calculate the mid point of list */
            mid = ((low + high) >> 1) - 1;
            i_ptr = *index_hndl + (mid *
                sizeof(index_entry));

            if (i_ptr->key > target)
            /* too big, so look in lower half */
                high = mid;
            else if (i_ptr->key < target)
            /* too small, so look in upper half */
                low = mid + 2;
            else
            /* found it, so store new key value */
                i_ptr->data = data;
        }

        /* extend handle by size of one entry */
        SetHandleSize(*index_hndl, (i_size + 1) *
            sizeof(index_entry));
        /* test for memory error here */
        n_ptr = i_ptr = *index_hndl + (mid *
            sizeof(index_entry));
        memcpy(++n_ptr, i_ptr, (i_size - mid) *
            sizeof(index_entry));
        /* do we insert as right or left? */
        if (i_ptr->key < target)
            i_ptr = n_ptr;
        /* and move in the new key and data value */
        i_ptr->key = target;
        i_ptr->data = data;
    }
} /* end of db_insert_key() */

```



System Sleuthing

World CDEV Investigated



Martin Minow
Mike Carleton
Arlington, MA
Volume 5, Number 9

Sleuthing the Map Cdev Martin Minow and Mike Carleton

[Martin Minow is a Principal Engineer at Digital Equipment Corporation. Originally a speech major (later a linguist), he decided he might need a “real” job someday, so he took the last programming course given for the Illiac I computer in 1962. When his graduate study grant ran out, he joined Digital Equipment Corporation in 1972, where he has held positions in software support and speech research and development, eventually working on the DECTalk speech synthesizer. He is currently a Principal Engineer in the mid-range system’s advanced development group. In his spare time, he runs marathons (slowly), and orienteers (even more slowly).

Mike Carleton received his BSEE in 1981. He first worked on special purpose compiler systems used to test the black boxes in fighter aircraft. After joining Digital Equipment Corporation in 1984 he supported the TOPS-10/20 COBOL compiler. Later he joined up as the system engineer for the VAX Supercomputer Gateway product and worked hand in hand with Cray Research engineers to make their two systems talk to each other. Currently, he is a Senior Software Engineer at Digital Equipment Corporation.]

System 6.0 includes an unexpected gimmick—a control panel program that displays a world map and allows you to select your location and time zone. The system is distributed with about 80 large cities already defined, and it is very easy to add your own location. However, there doesn’t appear to be any published way to extract the information for your own applications.

With a little poking around, we were able to decipher the internal information and think that other MacTutor readers might find it useful. By using the code in the following program, your phase of the moon program (or program to determine sunrise and sunset times) need not ask the user to specify the system’s location.

The program has only been tested on Mac SE’s and Mac II’s under System 6.0. At its heart is a subroutine to read data from the clock parameter ram.

The data is returned in three longwords with the following format (the actual numbers locate a suburb of Boston) found in figure 1.

Latitude and longitude are stored as a fraction of the circle. You can think of latitude and longitude either as signed quantities ranging from -.5 to +.5, or as unsigned quantities ranging from 0 to 1. The right method depends on your program’s needs. For example, since people don’t think of Boston (71°05’ West) as 288°55’ East, a program that displays the location would probably treat the numbers as signed, as shown in the example

1E	26	}	42° 24' North
AF	38		
CD	58	}	71° 14' West
5E	0E		
00	FF	}	5:00 West
B9	B0		

Parameter Layout

Figure 1

program.

Time is stored as the number of seconds offset from UTC (GMT) in the low three bytes of the third longword. The high byte is zero. When the 24-bit value is sign-extended, it will be positive for offsets East of UTC, and negative for Western offsets.

Sleuthing the Parameter RAM

The parameter RAM is a block of memory in the clock chip that is kept alive by a battery when the computer is turned off. It was included in the Macintosh to store a small amount of environmental information that should not change when the system is booted from a different floppy. The MAP CDEV data is stored here because Apple wanted your Mac to know that it is located in Boston even if it was booted from a disk created in Cupertino.

The classic Macintosh has 20 bytes of parameter memory. The contents are copied to low memory when the system is booted and you should access it by referencing the SysParam block as discussed in Inside Macintosh, volume II, chapter 13. When Apple introduced the Mac SE and Mac II, they used a clock chip with 256 bytes of memory. Since only the 20 bytes of the classic Mac are copied to system memory, we had to find a way to access the rest of the information.

The read_param_ram() function below is the C-language interface to the undocumented _ReadXPram trap that we found tucked away in the 256 KByte ROM. This routine can read any part (or all) of the 256 byte parameter RAM. It takes three parameters: a pointer to a buffer, an offset from the start of the parameter ram, and a count of bytes to copy to your program’s buffer. It returns an OSerr code which read_param_ram() then returns to your program.

Finally, we would remind you that you shouldn't assume that Apple will keep this information in the same place or format in future system releases.

In a recently-published document, Apple described ScriptManager 2.0 which contains support routines to allow reading and writing the Map information. As we show in the article, the information is stored in three longwords (Fract format) with the following format:

```
struct MachineLocation {
    Fract latitude;
    Fract longitude;
    union {
        char dlsDelta;
        /* Signed byte: daylight savings time delta */
        long gmtDelta;
    } gmtFlags;
};
```

The ScriptManager defines two functions to read and write the Map data:

```
pascal void ReadLocation(MachineLocation *loc);
pascal void WriteLocation(const MachineLocation *loc);
```

("const" is a keyword that was added to the Draft Ansi C Standard: it indicates that the parameter will not be changed by the function. You can omit it without harm.)

Latitude and longitude are stored as fractions of a great circle: see our code for details. The low-order three bytes of the third word (gmtFlags.gmtDelta & 0x00FFFFFF) contain the number of seconds the timezone is East of the prime meridian. Note that the high byte must be masked off and the high-bit propagated to get the timezone offset: this is shown in our program. The high byte is marked "reserved" but seems to be a repository for a Daylight Savings Time offset.

I don't know when the ReadLocation (and WriteLocation) functions will be available: they are not present in the current release of Think C. Until that time, our sample program should be usable, but the cautious program will switch to an approved interface as soon as possible.

```
/*
 * This is a simple program to demonstrate
 * extracting information from the Macintosh
 * parameter RAM and displaying the current
 * latitude, longitude, and time-zone offset.
 * Note that the format and manner of storage
 * of this data have not been published by
 * Apple.
 * This program is copyright © 1988, 1989
 * by MacTutor magazine. You may incorpor-
 * ate portions of this program in your
 * applications without restriction.
 * Compile using Think C, V3.0.
 *
 * Written by Martin Minow and Mike Carleton
 */
#include <stdio.h>

/*
 * Define the "Read parameter ram" trap and
 * the offset in the parameter ram where the
```

```
 * map is stored. This information doesn't
 * seem to be written down anywhere.
 */
#define _ReadXPRam 0xA051
#define Map_Offset 0xE4 /* Ram offset */

/*
 * This structure defines the location
 * currently set by the Map Cdev.
 * Latitude and longitude are long integers
 * specifying the fraction of the circle:
 * 1 degree East == 0x000308B9
 * 1 degree West == 0xFFCF747
 * All values are possible for longitude,
 * while latitude is constrained to ± 90°.
 * Only the low-order three bytes of timezone
 * are used. The value is the number of
 * seconds offset from UTC, East is positive,
 * West negative.
 */
#ifdef Use_Script_Manager
#include <ScriptMgr.h>

typedef struct {
    Fract latitude;
    Fract longitude;
    union {
        char dlsDelta;
        long gmtDelta;
    } gmtFlags;
} MachineLocation;

MachineLocation info;
pascal void ReadLocation(MachineLocation *);
#else
typedef struct {
    long latitude;
    long longitude;
    long timezone;
} MapDatum;

MapDatum info;
#endif

void main(void);
void dms(long, char *, char *);
void hms(long, char *);
OSErr read_param_ram(void *, int, int);

void printf(char *, ...);
int fgetc(FILE *);

void
main()
{
    OSErr status;
    /*
     * This is a test for divide rounding.
     * It should yield 180°00'00" W.
     */
    dms(0x80000000L, "Test", "EW");
    /*
     * Loop through here until the user types 'q' - this lets
     * you run the Map Cdev at the same time to see how location and
     * timezone changes affect the data.
     */
    do {
#ifdef Use_Script_Manager
        ReadLocation(&info);
        dms(info.latitude, "Latitude", "NS");
        dms(info.longitude, "Longitude", "EW");
        hms(info.gmtFlags.gmtDelta, "Zone");
#else
```



```

status = read_param_ram(&info,
    Map_Offset, sizeof (MapDatum));
if (status != noErr)
    printf("Error %d\n", status);
dms(info.latitude, "Latitude", "NS");
dms(info.longitude, "Longitude", "EW");
hms(info.timezone, "Zone");
#endif
} while (getchar() != 'q');
}

/*
 * Convert to degrees/minutes/seconds.
 */
void
dms(raw_value, what, zone)
long    raw_value;
char    *what;
char    *zone;
{
    register longvalue;
    register longdegree, minute, second;
    int    west;

    static longone_degree = (0x80000000L / 180L);
    static longone_minute = (0x80000000L / (180L * 60L));
    static longone_second = (0x80000000L / (180L * 60L * 60L));

    value = raw_value;
    degree = value / one_degree;
    value -= (degree * one_degree);
    minute = value / one_minute;
    value -= (minute * one_minute);
    second = value / one_second;
    if ((west = (raw_value < 0))) {
        degree = (-degree);
        minute = (-minute);
        second = (-second);
    }
    printf("%08lx: %3ld°%02ld'%02ld\" %c %s\n",
        raw_value, degree, minute, second, zone[west], what);
}

/*
 * Convert time to hours:minutes:seconds.
 */
void
hms(timezone, what)
long    timezone;
char    *what;
{
    register longvalue;
    register longhour, minute, second;
    int    west;

```

```

static longone_hour = (60L * 60L);
static longone_minute = (60L);

if (timezone & 0xFF000000) {
    printf("timezone high byte %08lx\n",
        timezone);
    timezone &= 0x00FFFFFF;
}
/*
 * Propagate sign bit from bit 23 to bit 31 if West of UTC.
 */
if ((timezone & 0x00800000) != 0)
    timezone |= 0xFF000000;
value = timezone;
hour = value / one_hour;
value -= (hour * one_hour);
minute = value / one_minute;
value -= (minute * one_minute);
second = value;
if ((west = (timezone < 0))) {
    hour = (-hour);
    minute = (-minute);
    second = (-second);
}
printf("%08lx: %3ld.%02ld.%02ld %c %s\n",
    timezone, hour, minute, second, "EW"[west], what);
}

/*
 * Read data from the clock parameter ram. The start parameter
specifies the offset within the parameter ram where the
read is to start. The count parameter specifies the number of
bytes to read.
 */
OSErr
read_param_ram(address, start, count)
void    *address; /* Result loc */
int    start; /* Ram start */
int    count; /* Read size */
{
    /*
     * Put the count into the high word of D0, the pRRAM start
address into the low word of D0, the Macintosh memory address
in A0, and trap to ROM. _ReadXPRam returns noErr on success
and prInitErr on failure.
     */
    asm {
        move.w    count,D0
        swap     D0
        move.w    start,D0
        movea.l    address,A0
        dc.w    _ReadXPRam
        return
    }
}

```



Advanced Mac'ing

Dot Matrix Printer Driver



Earle Horton
Hanover, NH
Volume 5, Number 10

A Dot-Matrix Graphics Printer Driver

This article describes a Macintosh printer driver for use with a Tandy DMP-110 dot-matrix printer. The printer resource file is written to be device-dependent, but it can serve as a skeleton for creating printer files for other devices. The routines to translate QuickDraw BitMaps to DMP-110 graphics codes are localized, so that one would only have to rewrite a few routines to create a driver for another dot-matrix printer. The printing code is in a combination of assembler and C, and may be compiled with either MPW C and Asm, v 2.0.2 or Aztec cc and as, v 3.6c. All files necessary for compilation with either system are included, but the Aztec-specific files assume that you have the MPW Shell, MPW Make, and Rez.

This driver implements the Print Manager high level printing calls, so that any Macintosh application which uses them can print on this printer. I attempt to emulate as closely as possible the ImageWriter driver in high-resolution (Best) mode, subject to the physical limitations of the printer. I provide portrait mode printing only, with a choice of scaling factors. Anyone who has taken a college math course will recognize the spirit of the following sentence: "I leave Landscape Mode printing as an exercise for the reader." The driver also implements two low-level procedures to dump BitMaps to the printer, so that command-shift-4 will work.

The printer file is missing some features which can be found in Apple printer files. It does *not* implement the driver's low-level BitMap proc or, indeed, *any* of the low level printing calls. Users of most applications will not notice this lack, since few applications use low level printing. It does not participate in text positioning and measuring or "line layout" as do Apple printer drivers. This means that although the printer file works well with screen fonts like Geneva and New York, LaserWriter fonts like Times and Courier sometimes give it problems. This is a feature which should be added, but it might take lots of code to do it right.

The code used in the printer file does not access low memory globals, except as these are used in library routines. It does not use any development-system-specific tricks to obtain "global" variables for the driver or for any of the code resources used in the driver. It does not use hard-coded numbers for the ioRefNum of the serial port or of the printer driver, but rather opens drivers by name. Any access to the Print Manager, e.g. to set or detect a printer error condition, is done through glue routines or through the PrGlue trap. In view of these precautions, the code *should* be fully capable of being used with other development systems, and *should* be compatible with future releases of Apple System Software, providing Apple does not change the interface to printer files too much.

The Printer File:

This section describes the resources you must create in order to implement a driver of this type. The structure of a printer resource file was described in my previous article (MacTutor Volume 3, Number 11 and 12), so I will give the resource list here, arranged roughly by function, and only describe the contents briefly. (You can get a list like this using MPW Rezdet with the "-l" command line option.)

DMP-110 Resource List

```
'PDEF' (0, Purgeable, Locked) # Code resources
'PDEF' (4, "Dialogs", Purgeable, Locked)
'PACK' (-4096, "Chooser Device", Purgeable, Locked)
'DRVR' (-8192, ".XPrint", Purgeable, Locked)

'HEXA' (-8192, "Printer Settings") # Port settings

'ICN#' (128, "DMP-110") # Bundle
'FREF' (128)
'Dmp1' (0)
'BNDL' (128)

'STR' (-4092, "Right Button") # For device package
'STR' (-4093, "Left Button")
'STR' (-4091, "List label")
'STR#' (-4080, "Baud Rates")

'DITL' (-8191, "Job dialog template") # Style and Job dialogs
'DITL' (-8192, "Style Dialog Template")
'DLOG' (-8191, "Job Dialog")
'DLOG' (-8192, "Style Dialog")

'DLOG' (-8190, "Next Page Box") # Miscellaneous
'DITL' (-8190, "Next Page Template") # dialogs and alerts
'ALRT' (-4079, "Sys41Alert")
'DITL' (-4079, "Sys41 Items")
'ALRT' (-4078, "ATalk is on!")
'DITL' (-4078, "ATalk Items")
'ICON' (-4080, "DMP-110")

'PREC' (0, "Default Scale") # Print Records
'PREC' (1, "Last Used Print Record")
'PREC' (2, "Shrink To Fit")
'PREC' (3, "Exact BitMap")
'PREC' (4, "Compatible")
```

Four code resources are included. The printer driver ('DRVR' -8192) and the dialog code ('PDEF' 4) are necessary to any printer resource file. At least one type of printing code ('PDEF' 0, here) is also necessary. The Chooser device package, ('PACK' -4096) is necessary if we wish to have adjustable settings which are not logically part of the standard Style and Job dialogs. A 'HEXA' resource is included to save global settings; it is quite small.

The Chooser device package uses both buttons, and displays

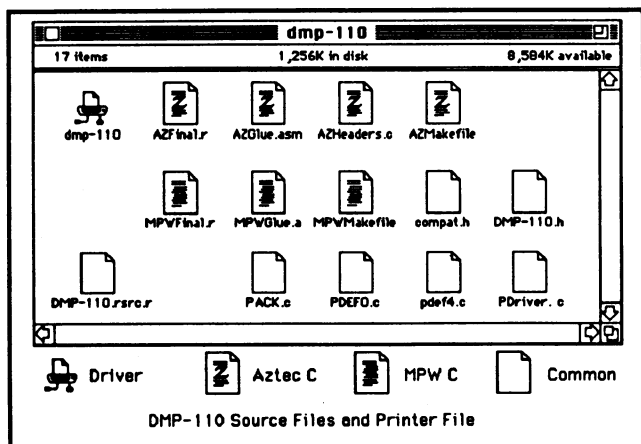
a string list of baud rate names in the Chooser's list box.

Four unique Print Record resources ('PREC') are used here, for four different printing styles. The same paper style is used in all four of them, but different scaling methods are used to obtain different printing effects. 'PREC' number 1 is used to store the last-used Print Record.

The rest of the resources are used in the dialogs and alerts, and the Bundle-associated information is used by the Chooser to display our printer file. Informative alerts and dialogs are provided for sheet feeding and for problem situations. An ICON resource containing the printer file icon is used to assure user identification of exactly who is putting up an alert. Alerts which might be displayed by the Chooser device package are given resource numbers which are in the legal range for the device package to use. Other Alerts and Dialog resources are given resource numbers belonging to 'PDEF' #0.

Cast of Characters: The Files

This section shows the source files used to build the printer resource file, and the icon used by my printer file.



The files with the 'Manx' icon ("Z") are used in creating the Aztec version of the printer file. The files with the MPW icon are for use with MPW C and Asm. The "Anonymous" files are for use with both systems. All are 'TEXT' files, with the exception of the printer file. Intermediate object, code and resource files are not shown.

Different source files exist for Aztec and for MPW for a number of reasons. Assembler syntax is different, so each system has its own "Glue" source file. Compiler and linker options, as well as mechanics of making the project, differ. Thus the two Makefiles. (Both use MPW Make, however.) The "...Final.r" file is different between the two systems because the Aztec linker refuses to create a non-'CODE' resource with ID zero, and it is more convenient to change this ID using the resource compiler. Finally, the file "AZHeaders.c" is used to create a precompiled header file for the Aztec C compiler. MPW C v 2.0.2 has no such feature.

Common files include two C language header files, a resource definition file, and four C language source files. The bulk

of the project code is in C; the assembler files provide necessary headers for code resources, as discussed in the 1987 article.

This project uses the Pascal interface to all ToolBox routines, as introduced in MPW C, version 2, and as described in Appendix H of the MPW C manual. An esthetic problem with this approach is that it requires the programmer to type in certain routine names in UPPER-CASE, e.g. GETNAMEDRESOURCE() for GetNamedResource(). The header file "compat.h" uses preprocessor macros to hide this ugliness. When using Aztec C, including "-D_INLINE" on the C compiler command line and using "sys2.lib;azc.lib" accomplishes the same thing. All ToolBox and Operating System calls used in this printer driver use parameters as described in Inside Macintosh as a result. Points are passed by value, unless they are listed as "VAR" parameters in IM. Strings are always Pascal strings. The MPW Pascal style interface does not appear to include all the List Manager calls, but this is taken care of in the C source files rather than in the header file. Perhaps MPW 3.0 high-level languages will have a uniform programming interface for all IM calls.

The C code as supplied can use either "PrintTraps.h" or "Printing.h" to define Print Manager calls which are used. For compatibility with systems older than 4.1, define BACKWARD_COMPATIBLE in "dmp-110.h" so that "Printing.h" gets used. Use of the calls in "PrintTraps.h" results in smaller code. Use of "Printing.h" assures compatibility with older systems. For a driver for your own use, then, the newer header file is appropriate if you are running all your Macs with System 4.1 or newer. For distribution, the older header file is probably better.

The header file "DMP-110.h" is the current version of "Daisy.h" from the 1987 article. It defines constants and typedefs which are shared between the various code resources. It gets smaller each time I work on it as the code moves closer to the point where the various code resources can be independent of each other.

The main resource definition file, "DMP-110.rsrc.r," is Rez source for all the non-code resources used in the printer file. If you are planning on using Aztec C or another system to build a printer file based on this one, and don't have MPW Shell, then you have to convert this file, or get the source disk and decompile the resources in the printer file.

This article includes Aztec listings, but all files in the folder picture will be included in the source disk.

The table entitled "DMP-110 Files Listing" shows all the files which exist in the project folder after a successful "buildprogram dmp-110" using the MPW language tools.

DMP-110 Files Listing

Name	Crtr	Type
AZFinal.r	Manx	TEXT
AZGlue.asm	Manx	TEXT
AZHeaders.c	Manx	TEXT
AZMakefile	Manx	TEXT
compat.h	????	TEXT
DMP-110	Dmp1	PRER
DMP-110.h	????	TEXT
DMP-110.rsrc	RSED	rsrc

DMP-110.rsrc.r	???? TEXT
Makefile	MPS TEXT
MPWFinal.r	MPS TEXT
MPWGlue.a	MPS TEXT
MPWGlue.a.o	MPS OBJ
MPWMakefile	MPS TEXT
PACK	RSED rsrc
PACK.c	???? TEXT
pack.c.o	MPS OBJ
PDEF0	RSED rsrc
PDEF0.c	???? TEXT
PDEF0.c.o	MPS OBJ
PDEF4	RSED rsrc
pdef4.c	???? TEXT
PDEF4.c.o	MPS OBJ
PDriver	RSED rsrc
PDriver.c	???? TEXT
PDriver.c.o	MPS OBJ

Messing Around in Assembler: The Glue

The specially assembled headers for the printing code resources are concentrated in one source file, and are now much easier to deal with. The MPW code is assembled in modules, using the PROC...ENDP directive pairs. All modules are kept in one object file. When the glue is linked with each C object file, the linker option “-m <module>” is used to determine the start address of the code resource. You have to name the glue file first on the linker command line, and have the entry point be the first module in the glue file which is picked up by the linker, in order to have the header appear at the top of the linked code resource. The structure of the assembler headers was discussed in the 1987 article, and is the same here.

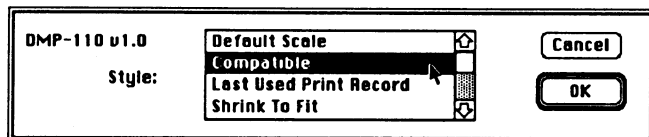
The Aztec glue is a little different, since the Aztec linker does not have a “-m” switch, and is not smart enough to pluck out the correct startup routine. Conditional assembly takes care of this. If you compile with Aztec, you just get four “.o” glue files instead of one. In order to prevent a conflict of entry points, the Aztec Makefile creates a special version of the runtime library with the default entry point removed.

The assembler glue defines some read-only constant data using “dc.b” and “dc.w” directives. This is because the MPW C compiler will not define string and array constants with PC-relative addresses. Since you have to have the glue anyway, why not make use of it? This is a convenient place to store device-dependent things like escape codes. Some C compilers do allow definition of string constants within source files which are to be linked into stand-alone code resources, but the trend appears to be to move away from this. Use of such also reduces portability.

The Aztec C libraries appear to be missing a glue routine for IsMPPOpen(). I have supplied assembler glue for this.

The Style Dialog: Image Scaling Strategies

This printer file includes a number of named ‘PREC’ re-



sources. A list in the Style Dialog allows the user to choose between them. Only one paper size is currently allowed, but a number of scaling factors are supplied. Each style is appropriate for a particular kind of application program or printing effect. This is a convenient dialog to use during development, since the dialog code doesn’t have to know a lot about the printer’s capabilities, and you don’t have to recompile to try out a different ‘PREC’. The printing code figures out at run-time how big the paper is, and how to scale the image in the Printing Port before sending it to the printer. If you design a printer driver which allows a number of different options which are independent of each other, then a more elaborate dialog box might be appropriate.

When one of the list cells is selected and the dialog is confirmed, the dialog code copies the appropriate fields from the ‘PREC’ resource whose name matches the selected cell. The dialog code also records the resource number of the chosen ‘PREC’ in one of the private fields of the Print Record which it returns to the application, and uses this information to choose the style to select when putting up a new Style Dialog. This means that the last-chosen style for a document is “remembered” if the application saves the Print Record.

Most of the Print Records supplied with this printer file specify Picture-scaling to enlarge the printed image before sending it to the printer. In order to understand the effect of scaling on the appearance of the printed output, it is necessary to understand the differences which exist between the capabilities of applications to cope with scaling.

Paint programs print images strictly as BitMaps. Whatever you have on the screen is mirrored, bit for bit, in the printing GrafPort when using one of these programs. These usually cannot scale images themselves, nor do they produce images which can be scaled attractively. HyperCard is a perfect example.

Font-oriented programs, such as word processors, provide the ability to adjust the spacing between characters when printing justified text. This may cause problems when we try to print an image at a different resolution from that in which it has been drawn originally.

For scaling to be successful, a larger size font must exist, or the text will be printed in a scaled font, which will not be attractive. If we scale an image containing justified text by a non-integral factor, then quantization errors in text positioning cause justification to fail. This is a problem with the DMP-110, since there is no way to divide 72 dpi (screen) evenly into 120 dpi (paper). A way around this is to provide a Printing Port resolution of 60 dpi, and scale by 2, but this means the apparent size differs from screen to paper, and results are not really WYSIWYG.

Another problem arises when we scale by an integral amount, and the sizes of characters in the font family are not linear. For example, the capital ‘A’ in Times 12 has a width of 8, but the capital ‘A’ in Times 24 has a width of 17. If the character sizes are not linear, then justification fails when the Picture is enlarged, even if by an integral amount. Screen fonts like Geneva always print with justification preserved if the output Picture is scaled by an integral amount, because character

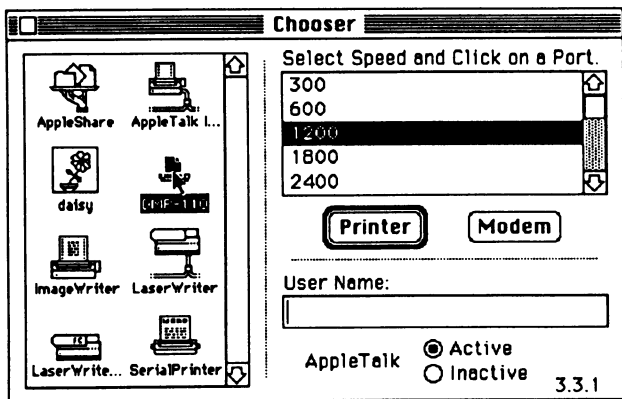
sizes are linear. LaserWriter fonts, on the other hand, seldom print correctly under these conditions, because character sizes are *not* linear. The printer file needs to exchange some information with the Font Manager in these cases, so that non-linear fonts will be positioned correctly. Since I don't do this, my printer file will not print LaserWriter fonts so that the right ends of justified lines are properly lined up.

The most "advanced" printing application type is a program which handles images as a collection of objects. The size of an object is determined using non-screen units such as inches or centimeters, and the program can handle practically any printer resolution you throw at it. Since these programs do not store images as collections of bits, but rather as drawing instructions, they are (should be) able to draw an object to the correct scale no matter what the printer resolution or paper size. The only real limitation here is that, again, scaling of text is limited by the collection of installed fonts. Examples are MacDraw and similar object-oriented drawing programs, CAD programs, and VersaTerm PRO when printing TekPrint files.

The Print Records provided by this dialog provide dot-addressable graphics for BitMap and drawing programs, a "Compatible" or near WYSIWYG mode, and a scaling factor which preserves text justification while allowing use of screen fonts, but not all three at once. With another printer and a more ImageWriter-like resolution, you may do better. Remember that the numeric values stored in my 'PREC's will not produce attractive results with your printer, unless it has the same resolution as mine. The comments in the resource definition file show what I try to achieve with each Print Record.

A certain amount of experimentation is necessary to determine what the Style Dialog should put in the Print Records, in order to achieve any special effects which your printer is capable of. I recommend using a simple interface such as that above while you are experimenting with printing styles, and then move to a more elaborate, customized dialog when you have determined what kind of results you get from your printer and your Macintosh.

The Device Interface: Talking to the Chooser



The file type for this printer file is 'PRER', so that we can use a Chooser Device Package interface to solicit configuration information from the user. The Chooser's list is used to put up

a list of baud rates, and the two Chooser buttons are used to select which serial port to use. Because this printer file is written to be device-dependent, there are no further options which may be set from the device package. Device-dependent parameters like escape codes are not changed by the user.

When the user clicks one of the buttons or list items, the device package would like to communicate the new settings to any currently open printer driver which belongs to us. This is implemented here simply by changing the 'HEXA' resource and writing it out to the resource file. Since this printer file implements high-level printing jobs and screen dumps only, each printing request is treated as a separate print job, and the new settings will be picked up before any more printing is done. If you decide to implement low-level calls, then you have to figure out a way to communicate settings changes from the device package to the driver. This is not easy.

The device package checks for a system version of 4.1 or newer, and puts up an error Alert if the device is selected, the driver was compiled to use "PrintTraps.h," and the system is too old. If AppleTalk is active and the printer port button is clicked, then the device package informs the user that this is an impossible choice. (There is no way to dim the left button, except by emptying the Chooser's list.)

The Driver:

This driver is similar in structure to the driver described in my previous article, but it does have some significant enhancements. The driver does not use any development-system-specific tricks to maintain private storage. Each active routine in the driver performs a single function, then cleans up after itself when done. Support for low-level printing functions is removed.

The driver *only* does screen dumps. The screen dump routines use essentially the same algorithms that are used by the high-level 'PDEF' code, so I won't discuss them in any detail. They will, however, have to be changed if your printer accepts different graphics codes from the ones mine uses. The hardest task is to find the Rect which encloses the top window, since the portRect doesn't include the structRgn of a window. If anyone has a better method of finding this Rect, I would be interested in hearing from you.

The Open, Close, and Status routines simply return noErr. The Control routine opens the chosen serial port, dumps the screen or top window if called for, closes the serial port, and logically should return an error code if an error occurred during printing. If the control routine does return an error code, however, the 'FKEY' which calls it for a screen dump just tosses the error code in the byte bucket, assumes that the driver was not open, opens it, and calls the control routine again. For this reason, this driver control routine always returns noErr. The driver has no global storage, doesn't do anything with its device control entry, and is in general quite friendly to other parts of the System.

Coping with Errors: Making the Best of a Bad Time

This driver does much more complete error checking than the driver in my previous article did.



This printer cannot use the printer port now because AppleTalk is active. Please either turn off AppleTalk, use the modem port instead, or choose another printer.

OK

My driver is a 'PRER' device, so it gets to use the Chooser Device Package interface for such things as baud rate selection. This also means it does not get the "automatic" serial port selection which the Chooser does for 'PRES' devices like the ImageWriter. A common user error is to turn on AppleTalk when the driver has been configured to use the printer port. When printer driver is opened it finds AppleTalk on, but its configuration resource tells it to use the printer port. The Alert box above tells the user that this has happened, and the driver closes itself. Other open errors are possible, but much less common. These are handled by simply closing the driver, and returning an error code.

Aside from errors which may occur during the driver's open routine, there are three types of error which I look for in the 'PDEF'. First, the user may type command-'.' at any time during printing. The routine pointed to by the prJob.pIdleProc in the print record is called often during high-level printing, and sets PrSetError() to other than noErr when this happens. (The default routine which I supply now uses iIOAbortErr instead of iPrAbort for this purpose.) The printing code allocates temporary output buffers to use when translating BitMaps, and sets PrSetError() to memFullErr if NewPtr() ever returns nil. Finally, errors may occur during a write attempt to the serial port. Serial port errors are rare, but should be anticipated.

Another possible error is that free memory might run out when the application is drawing into the Printing Port, and the system is filling a PicHandle with drawing commands. The only thing I do about this is hope that it doesn't happen!

The procedure for handling aborts in printing 'PDEF's and in the driver is discussed below.

The top-level routine calls setjmp() before sending any characters to the printer, and stores the jmp_buf[] array and IO Parameter block in a data structure which is passed down the calling sequence. All serial port output is done through the routine asyncwrite(), which takes a pointer to the data structure and a pointer to an idle routine as parameters. Asyncwrite() issues an *asynchronous* write request to the serial driver, and repeatedly calls the idle routine until the request has been completed. (The ioResult field of the parameter block used to issue the write request will remain positive until the write request has been completed.) If a user abort (command-'.') occurs, then the idle routine calls PrSetError() with iIOAbortErr. (The default idle routine I include does this, but an application-supplied routine may arrange for printing to be halted in another way.)

After the idle routine returns, asyncwrite() checks PrError() to see if an abort or error has been logged.

If asyncwrite() detects that an abort or error has occurred during the write, it first calls PBKillIO() to kill any pending serial port output, then calls longjmp() to get back to the top level printing routine. The top level of the control routine does any necessary cleanup, and then calls PrSetError(iIOAbortErr).

When the write request is complete, asyncwrite() checks the ioResult field of the IO Parameter block for the presence of an error code, and longjmp()s back to the main level if it finds one.

Since a printer is a slow device, it is important to detect errors quickly, particularly user abort errors. A user abort may mean a mechanical problem with the printer, so we want to detect it and kill any pending output at the earliest possible moment. This method provides that ability, but it does have one problem with the DMP-110 printer. High resolution graphics are sent to the printer by first sending an escape sequence telling the printer to expect a number of two-byte codes between 1 and 959, then sending the codes. If the printing code is sending a sequence of graphics codes, all of which are non-zero, then the codes will be sent all at once. This can happen if the image contains a horizontal line, for instance. If an abort is sensed in the middle of a long sequence of graphics codes, then the printer will be left hanging, expecting perhaps several hundred more graphics codes. Since all characters are valid graphics codes in high resolution mode, there is no way to get the printer out of this state, short of shutting it off and turning it on again. This may cause the user some unpleasantness by leaving the printer in an undefined state, but perhaps less than if the driver tries to print to the end of the line, while the ribbon or paper is jammed. If your target printer has a means of resetting itself with a serial port command at this time, you would want to send the reset command before the control routine returns.

A possible source of problems under MultiFinder is that the printer driver is a shared resource, and so it is highly unlikely that the printer driver's close routine will ever get called when an application exits. This means that the serial port might get left open, and the user can't use it for something else. For this reason, the driver doesn't maintain any storage, and closes the serial port before it returns. Each screen dump or high-level printing job is handled separately, and everything is cleaned up afterwards. This means that if the user changes printers, or wants to use a terminal program on the same serial port after she is done printing, then she won't have any problems doing so.

Finally, there is one more source of errors, if the programmer decides to use "PrintTraps.h" instead of "Printing.h." This is best explained by the the Alert box:



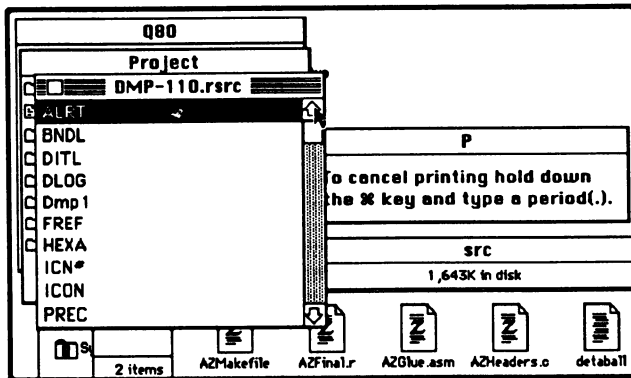
This Printer requires System 4.1 or newer!

OK

My copy of the printer file uses "PrintTraps.h" because (a) it is strictly for my personal use, and (b) I use the newest system file I can get at all times. If you want to write a printer file for commercial use, then "Printing.h" is highly recommended. Using "Printing.h" adds about 2k to the size of the complete printer file. If you plan to write a printer file for a color device which uses Color QuickDraw, or need any features in System 4.1 for any other reason, then you might as well go ahead and use "PrintTraps.h."

Background Printing: Working with MultiFinder

The default idle procedure used when high-level printing is active calls WaitNextEvent() if the trap is implemented, and SystemTask() and GetNextEvent() otherwise. This allows a user to put a printing application into the background under MultiFinder, and allows desk accessories to continue running under



UniFinder. Combined with the asynchronous method of writing to the serial port, this gives the user use of the machine when the serial driver is waiting to send characters. In the screen dump above, the Finder is printing the directory of the "src" folder to the DMP-110 printer, and ResEdit is running in the foreground. Performance of ResEdit is just fine under these conditions. This is not the scheme that Apple has chosen to use for background printing, but pending *documentation* of Apple's scheme it is an acceptable alternative.

One problem with this scheme is that most applications will put up a dialog box when printing similar to the Finder's, above. MultiFinder doesn't like to swap out an application when the front window is a dBoxProc type of window. Changing the procID of the Finder's printing dialog box to documentProc allows the user to freely swap the Finder in and out of the foreground while printing, and the Finder doesn't know a thing about it! Setting the "Can Background" bit in the application's 'SIZE' resource is also necessary to an older application to be given background time if you swap it out while printing.

Printing speed seems to be about the same under these conditions as it is with the printing application in the foreground or operating without MultiFinder. With asynchronous writes, and the limiting factor being the speed of the print head, there is usually a lot of free time available when printing.

The driver's procedure for dumping the screen does not call WaitNextEvent(), for the simple reason that allowing the user to move windows around would really muck up the output!

BitMap to Dot-Matrix: Nuts and Bolts

This section shows how I transform a QuickDraw BitMap to a series of graphics codes which *my* printer can understand. It is necessarily device-dependent. Your output format may vary, depending on the characteristics of your printer, but your input will be the same if you are talking to a black-and-white printer. Writing a printer file for a color printer is "left as an exercise for the reader."

The DMP-110 accepts high-resolution printing codes in the following format. A two-byte escape code is used to signal high-resolution graphics. This is followed by another two-byte code which tells it how many columns to print. Each 16-dot column is specified by a two-byte code. In order to print *n* columns of graphical data we send the two-byte graphics code, two-bytes to specify *n* columns, and then *n**2 bytes of graphical data. There are 959 possible 16-dot columns on one line, for a printable width of just under eight inches.

The first byte specifies the top eight dots, and the second specifies the bottom eight dots. The low-order bit in the first byte is on if the top dot in the first column is to be printed, off otherwise. The high-order bit in the first byte is responsible for the eighth dot in the first column. The second byte in the two-byte sequence accounts for the ninth through sixteenth dots in the first column. This is continued until 2*n bytes are sent, with the last byte specifying the bottom eight dots in the last dot-column. There is also a four-byte sequence for positioning the print head at any of the possible 959 printing positions.

Because of the way the 68000 series processors store numeric quantities in RAM, the first or top dot corresponds to the eighth bit in the first byte sent, which is the least significant bit. This is very important to remember because the Toolbox Utilities routines which are used to move bits around access them in the opposite order. In order to copy a bit from a BitMap into a dot-position in the first or any other column, we have to take into account the logical bit-reversal which occurs during this process. A sixteen pixel deep Rect in the source BitMap is copied to an output buffer for the DMP-110 in the following fashion.

A double loop loops over the dot-columns to be printed, up to 959 in number. Two inner loops fill the top and bottom bytes with bits from the source BitMap. A QuickDraw BitMap consists of bytes which are laid on their sides, so that the first pixel in the first row is specified by the first bit in the data pointed to by the baseAddr field of the BitMap. This bit is mapped to the least significant bit in the first byte in the output buffer. In the scheme used by the Toolbox Utility routines, the first bit in the source BitMap is mapped to the *eighth* bit in the output buffer. The seventh bit in the first output byte comes from the first bit in the second row of the source BitMap. The first inner loop fills the first output byte with the first bits from each of the first eight rows in the source BitMap, and the second inner loop uses the first bits from the ninth through sixteenth rows of the BitMap to fill the

second output byte. The outer loop continues until all of the dot-columns in one 16-dot high printing row have been filled.

Landscape style printing should be easier to implement. If the BitMap is scanned in the correct direction then bits within bytes will not have to be moved around, but only whole bytes.

A number of schemes may be used for sending the output buffer to the printer at this point. The entire row of graphics codes might be sent at one time, simplifying the computation. If, however, the output page will contain many white areas, it is advantageous to reposition the print head during printing so as to skip over white areas. The DMP-110 requires four bytes to position the head, and two bytes to print a single blank column. Accounting cost in terms of number of bytes sent through the serial port, if we skip over three blank columns we win, if we skip over two columns we break even, and if we skip over one column, we lose. How to optimize the printing process is harder to figure out for a whole row, but it is probably correct to assume that white columns will be grouped together, and that some improvement is to be expected by skipping over any blank dot-column. The printing head is then positioned before the next non-blank printing area. More improvement might be expected by skipping over only blank dot-columns where they are grouped, but at the expense of having to write more code.

A BitMap is printed in this scheme by first breaking it up into areas which are sixteen pixels deep. Each such area is converted and sent to the printer, followed by a carriage return and a high-resolution graphics linefeed. By looping over the whole BitMap, an exact image of the BitMap is made on paper.

This general scheme will work for any dot-matrix printer which accepts byte codes to specify a dot-column, if it is first modified to use the device-dependent protocol needed by the printer. Changes to this source code to work with another printer should involve changing escape codes and reworking the BitMap translation routines only.

First, look in the printer manual to determine how many vertical dots are printed in each dot-column. This determines how many rows of the source BitMap may be printed at once. Second, find out how the dots in a column are encoded into bytes to be sent to the printer. This will most probably be top-dot-first using Intel or VAX ordering. Third, map the bits in the BitMap to a suitable output buffer. Fourth, and this is optional but recommended, determine whether optimization of output byte codes to take advantage of repositioning the print head is possible. Fifth, send the printing codes to the printer along with the proper device-dependent escape codes and a graphics line feed. Finally, if the printer is capable of using form feed codes to separate pages, send the form feed at the end of the page. Otherwise, advance the paper using more line feeds.

BitMap to Band: Capturing the Image

This section details how to trick an application program into drawing a picture in our printing GrafPort, how to "play back" the picture in bands using DrawPicture(), and how to send the bands on to the BitMap translation routines. In order to write a printer file for another device, the size of the BitMap, the size of the

GrafPort's portRect, and the number of bands will have to be tailored to your printer. The general principles remain the same, and I believe that the method I use here is a good general purpose method for capturing QuickDraw images to be sent in high-resolution mode to any dot-addressable device.

My code does not take advantage of a printer's native fonts, since all information is printed in graphics mode. It relies on QuickDraw and on Macintosh screen fonts for all text printing. How to combine graphics and use of a printer's native fonts is left as "an exercise for the reader" or perhaps a future article.

The first routine to be called in a printing job is PrOpenDoc(). This routine sets up the fields of a Printing GrafPort (TPrPort) to match those in the Print Record passed to it, allocates necessary storage, and initializes variables used in the printing job. The IGParam[1-4] fields of the TPrPort are used to hold all printing variables particular to this printing job, and dynamic storage is allocated in a tree structure branching from pointers stored in these fields, which are given preprocessor names for readability. This routine allocates a total of about 3k of storage. If the storage cannot be found, then it calls PrSetError(memFullErr), and leaves things in a state where PrCloseDoc can clean up properly. The application will call PrCloseDoc if there is an error logged after this routine returns.

Use of distinct blocks of storage to hold the printing variables complicates the process of allocating and disposing of them, but allows for the use of an output BitMap whose size is determined at run time.

The dimensions of the TPrPort are taken from the prInfo.rPage field of the Print Record, and the dimensions of the port's BitMap are taken from the prXInfo field. The bounds Rect of the BitMap is sized so as to be able to break up the portRect into several "bands" used in printing. These must agree with the physical capabilities of the printer. The serial port is opened for this printing job, and a pointer to the TPrPort is returned to the caller.

The PrOpenPage() routine is called before every page to be printed. It sets the port to the Printing Port, and opens a new picture. All subsequent drawing in this port is saved in the picture, whose handle is saved in one of the IGParam fields of the Printing Port.

The PrClosePage() routine does all the work of translating and sending the saved picture to the printer. It first calls ClosePicture to finish the picture definition. Then the portRect of the Printing Port is resized according to the information in the prInfoPT field of the Print Record, so as to reflect the final page size. This information is taken from the Print Record, and not from the code, to allow for easier adaptation to other printers. Note that you don't have to *print* the picture at this point, you can do anything you want to with it. You could for instance save the images printed by an application in a picture file or a paint file.

Printing is accomplished as follows, for each band in the printing page:

The Printing Port's BitMap is first aligned with the band to be printed, using the MovePortTo() Toolbox call. The Printing Port's portRect is erased. DrawPicture() is called to draw the application's picture in the expanded portRect, thus correctly

filling the bits in the BitMap for the band to be printed. The routine dumpbits() is then called to convert and send the band to the printer.

This process is repeated until the entire picture has been printed, and then KillPicture() disposes of the picture Handle.

Since this is the routine which communicates with the serial port, and it also calls DrawPicture() several times, this is the routine wherein the printing code spends the most time. The prJob.pIdleProc routine should be called repeatedly whenever the printing code is waiting for the serial driver to finish a write request. This is done in the routine asyncwrite(), which also checks serial port errors. If this routine ever senses an abort attempt or a serial port error, then it longjmp()s back to the top of PrClosePage, which calls PrSetError() and returns.

When the last page has been printed, then the application calls PrCloseDoc, which closes the serial port and the Printing Port, reclaims all dynamic storage that was successfully allocated, and exits.

Putting It All Together: Compiling and Stuff

Compiling the sources is perfectly straightforward, if you have the MPW Shell and either MPW C and MPW Asm, or Aztec cc and Aztec as. Copy the files from the source disk into a new folder, copy either "MPWMakefile" or "AZMakefile" to "Makefile," and type "BuildProgram install." This builds the driver, and installs a copy of it in the active System Folder. If you are using Aztec C, the script generated by BuildProgram will ask you to find an original copy of "sys2:lib:azc.lib." This is copied to "{clib}tools.lib" and modified for use with this project, by deleting the module containing crt0.o. This step is necessary so that the linker will use my assembler glue as the entry point, and not crt0. Also make sure that when using Aztec cc, your {INCLUDE} environment variable points to the directory containing the MPW-compatible header files, and not the older Aztec header files.

If you do not have the MPW Shell, and want to build the printer file using Aztec Make, then you will have to modify AZMakefile to use Aztec Make syntax. Since I purchased the "cheap" Aztec C package, I did not get Aztec Make with it. If you also bought the "cheap" Aztec package, but do not have MPW Shell, then you will have to write a script to compile and link the files, or (ugh!) type in the commands by hand.

Also, MPW users might want to go through their "Printing.h" and "PrintTraps.h" files and replace every single instance of the keyword "int" with "short." This has nothing to do with this project, but it will allow you to call PrGeneral from other programs. Presumably, this problem has been fixed in MPW C v 3.0.

Results: What Does it Look Like?

It is difficult to include samples of printed output, since MacTutor probably uses a page setup program, and would have to scan output I sent them and then paste it back into the final article before printing. We can only hope.

The most serious flaw with the choice of printer is that the resolution is not ImageWriter-compatible. This is a hardware limitation of the device I am using, and may not affect your efforts.

The main shortcoming of this printer file is that it does not participate in line layout at all, and thus cannot reproduce justified text with LaserWriter fonts if the output image is scaled. I haven't begun to investigate what is required to do this properly, and so I don't know how much work is involved in implementing this feature. Images containing linear screen fonts like Geneva, New York, and Symbol always seem to come out right if the scale factor is an integer.

Bit-mapped images do not produce the same size printed output as would the ImageWriter or the LaserWriter in ImageWriter-compatible (default) mode. Paint images which are scaled by an integral factor come out bigger or smaller than they look on the screen as a result, but all the bits are there, and in place. You will not have this problem if your printer is capable of 72 dpi, or a multiple thereof.

Word processor programs produce acceptable output if a screen font exists in the correct size for QuickDraw to use when DrawPicture() is called. For example, if a 9-point font is used, the image is doubled in size before printing, and an 18-point font can be found whose character sizes are twice those of the 9-point font, then text is positioned properly on the output, and justification works properly. More work is necessary to make the printer file work properly in all cases with LaserWriter fonts and others with non-linear character sizes.

Drawing programs like MacDraw work the best with this printer file, since these do the work of scaling by themselves. With a drawing program, one uses a Print Record which calls for no scaling on the part of the printing code, and the drawing program can then place and size objects to best advantage. These programs look in the prStl field of the Print Record to determine the paper size, and thus the correct size of the printed object. Such programs are capable of producing identically-sized output with my printer and my driver as they do with Apple printers and Apple drivers.

Although this printer file has some shortcomings, it *does* allow printing of Macintosh graphics on my DMP-110, which had been gathering dust since I stopped using my Radio Shack Color Computer. Results are in most cases attractive, and it suffices for most of the draft graphics printouts I need. The missing features are left for the subject of a future article, or perhaps an exercise for the reader.

Listing: MPW MakeFile

```
# file is part of DMP-110 printer driver for the Macintosh
# series of computers.
# Earle R. Horton Wednesday, November 30, 1988
# Makefile for DMP-110 v1.0.
# This is the Makefile for use with MPW C, v 2.0.2.
CFLAGS = -g
.c.o f .c
    C (CFLAGS) (default).c -o (default).c.o
.a.o f .a
    Asm (default).a -o (default).a.o
DMP-110 ff PACK PDEF0 PDEF4 PDriver DMP-110.rsrc mpwfinal.r
Rez -o DMP-110 -c 'Dmp1' -t 'PRER' mpwfinal.r
```

```

SetFile DMP-110 -a B
DMP-110.rsrc ff DMP-110.rsrc.r makefile
rez -o DMP-110.rsrc -t 'rsrc' -c RSED DMP-110.rsrc.r
PACK ff MPWGlue.a.o pack.c.o
link MPWGlue.a.o pack.c.o
-m PACKENTRY -d
(libraries)interface.o -rt PACK=-4096 -sn
Main="Chooser Device"
-ss 100000 -o PACK -t 'rsrc' -c RSED
PDriver ff MPWGlue.a.o PDriver.c.o
link -m DriverEntry -d MPWGlue.a.o PDriver.c.o
(cibraries)cinterface.o
(libraries)interface.o (cibraries)CRunTime.o
(cibraries)StdCLib.o
-rt DRV=-8192 -sn Main=.XPrint
-ss 100000 -o PDriver -t 'rsrc' -c RSED
PDEF0 ff MPWGlue.a.o PDEF0.c.o
link -m PRINTENTRY -d MPWGlue.a.o PDEF0.c.o
(cibraries)cinterface.o
(libraries)interface.o (cibraries)CRunTime.o
(cibraries)StdCLib.o -sn Main="Draft Printing Code"
-rt PDEF=0 -ss 100000 -o PDEF0 -t 'rsrc' -c RSED
PDEF4 ff MPWGlue.a.o PDEF4.c.o
link -m DIALOGSENTRY -d MPWGlue.a.o PDEF4.c.o
(cibraries)cinterface.o
(libraries)interface.o -sn Main=PrDialogs
-rt PDEF=4 -ss 100000 -o PDEF4 -t 'rsrc' -c RSED
install f DMP-110
duplicate -y DMP-110 "(systemfolder)DMP-110"
clean f
delete -i "(systemfolder)testDMP-110"
files -m 300 -t rsrc ;files -m 300 -t PRER;files -m 300 -
t 'OBJ'
PDEF0.c.o f DMP-110.h
PDEF4.c.o f DMP-110.h
PDriver.c.o f DMP-110.h
pack.c.o f DMP-110.h

```

Listing: compat.h

```

/* FILE is intended for use with MPW C 2.0 Interface files
 * which generate InLine code for routines that call the
 * ToolBox with points by value & strings as Pascal strings.
 * It should be included AFTER any Macintosh #include files.
 * See Appendix H of the MPW C 2.0 manual for details.
 * file allows use of MPW C 2.0 standardized ToolBox calls
 * using spelling from Inside Macintosh, and not all upper
 * case, as is used in MPW C 2.0 header files. Please note
 * that use of this means that you no longer have access
 * to the C-language versions of the ToolBox calls.
 * It is rumored that this file will no longer be necessary
 * with MPW C 3.0. (?)
 * If you like to pass C strings to ToolBox, do NOT include
 * this FILE. */
#ifdef MPU68000 /* Aztec C, v 3.6c. */
/* -D_INLINE on cc's command line is all that's required. */
#else
#define NewControl NEWCONTROL
#define SetCTitle SETCTITLE
#define GetCTitle GETCTITLE
#define DragControl DRAGCONTROL
#define TestControl TESTCONTROL
#define TrackControl TRACKCONTROL
#define FindControl FINDCONTROL
#define OpenDeskAcc OPENDESKACC
#define FindDItem FINDITEM
#define NewDialog NEWDIALOG
#define ParamText PARAMTEXT
#define GetIText GETITEXT
#define SetIText SETITEXT
#define NewCDialog NEWCDIALOG
#define GetVol GETVOL
#define SetVol SETVOL
#define UnmountVol UNMOUNTVOL
#define Eject EJECT

```

```

#define FlushVol FLUSHVOL
#define Create CREATE
#define FSDelete FSDELETE
#define FSOpen FSOPEN
#define OpenRF OPENRF
#define Rename RENAME
#define GetFInfo GETFINFO
#define SetFInfo SETFINFO
#define SetFLock SETFLOCK
#define RstFlock RSTFLOCK
#define GetFontName GETFONTNAME
#define GetFNum GETFNUM
#define LCellSize LCELLSIZE
#define LClick LCLICK
#define LNew LNEW
#define NewMenu NEWMENU
#define AppendMenu APPENDMENU
#define SetItem SETITEM
#define GetItem GETITEM
#define InsMenuItem INSMENITEM
#define MenuSelect MENUSELECT
#define EqualString EQUALSTRING
#define RelString RELSTRING
#define UpString UPSTRING
#define DIZero DIZERO
#define NumToString NUMTOSTRING
#define StringToNum STRINGTONUM
#define SFPutFile SFPUTFILE
#define SFPPutFile SFPPUTFILE
#define SFGetFile SFGETFILE
#define SFPGetFile SFPGETFILE
#define IUDateString IUDATESTRING
#define IUDatePString IUDATEPSTRING
#define IUTimeString IUTIMESTRING
#define IUTimePString IUTIMEPSTRING
#define IUCompString IUCOMPSTRING
#define IUEqualString IUEQUALSTRING
#define DIBadMount DIBADMOUNT
#define DrawString DRAWSTRING
#define StringWidth STRINGWIDTH
#define StuffHex STUFFHEX
#define AddPt ADDEPT
#define SubPt SUBPT
#define EqualPt EQUALPT
#define PtInRect PTINRECT
#define Pt2Rect PT2RECT
#define PtToAngle PTOANGLE
#define PtInRgn PTINRGN
#define StdText STDTEXT
#define CreateResFile CREATERESFILE
#define OpenResFile OPENRESFILE
#define OpenRFPPerm OPENRFPERM
#define GetNamedResource GETNAMEDRESOURCE
#define GetINamedResource GETINAMEDRESOURCE
#define GetResInfo GETRESINFO
#define SetResInfo SETRESINFO
#define AddResource ADDRESOURCE
#define GetAppParms GETAPPPARMS
#define OpenDriver OPENDRIVER
#define TEGetOffset TEGETOFFSET
#define TEGetPoint TEGETPOINT
#define TEClick TECLICK
#define NewString NEWSTRING
#define SetString SETSTRING
#define GetIndString GETINDSTRING
#define DeltaPoint DELTAPoint
#define ShieldCursor SHIELDCURSOR
#define NewWindow NEWWINDOW
#define SetWTitle SETWTITLE
#define GetWTitle GETWTITLE
#define NewCWindow NEWCWINDOW
#define GrowWindow GROWWINDOW
#define DragWindow DRAGWINDOW
#define TrackGoAway TRACKGOAWAY

```

```

#define FindWindow FINDWINDOW
#define PinRect PINRECT
#define DragGrayRgn DRAGGRAYRGN
#define TrackBox TRACKBOX
#endif

```

Listing: DMP-110.h

```

/* Earle R. Horton.
 * Wednesday, November 30, 1988
 * All rights reserved.*/
#ifndef __DMP__
#define __DMP__
/* Define BACKWARD_COMPATIBLE to use printing glue,
 * rather than _PrGlue trap. Penalty is about 2k size in
 * final printer resource file.*/
#ifdef BACKWARD_COMPATIBLE
#include <Printing.h>
#else
#include <Printtraps.h>
#endif
#include <setjmp.h>
#include <Devices.h>
#include <AppleTalk.h>
#include <Files.h>
#include <Serial.h>
#define PrintErr (*(short *)(&0x0944)) /* Not used if PrGlue
trap found. */
#define SHEETDIALOG (-8190)
#define DONEITEM 1
#define STOPITEM 2
#define RES1ID (-4080)
#define RES2ID (-8192)
#define SYSNEEDED (0x0410)
#define ATALKALERT (-4078)
#define SYSVERSALERT (-4079)
#define iPrRelease 3
#define GRAFREZZ 120
#define NORMALREZ 80
#define COMPATREZ 72
#define VERSION 3
#define iFMgrCtl 8
#define iPrEvtCtl 6
#define lPrLFEighth 0x0003FFFE
#ifndef FALSE
#define FALSE 0
#define TRUE 1
#endif
#define iPrPgMax 9999
#define iPrPgFst 1
typedef struct {
    struct QElem *qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    ProcPtr ioCompletion;
    OSErr ioResult;
    char *ioNamePtr;
    short ioVRefNum;
    short ioCRefNum;
    short csCode;
    long lParam1;
    long lParam2;
    long lParam3;
} PrParam, *PPrParam;
typedef struct {
    short dummy[5];
} pconfig, *Pcfg, **Pfg;
#define pport ((*settings)->dummy[0])
#define pbaud ((*settings)->dummy[1])
#define XonXoff ((*settings)->dummy[2])
#define MAGIC 'Dmp1'
#define PORTOPEN 'OPEN'
#define dOpened 5
typedef struct {

```

```

ParamBlockRec iopb; /*writing to serial driver. */
jmp_buf abortbuf;
Ptr obuf;
}Dstorage,*DPstorage,**DHstorage;
#endif

```

Listing: MPWGlue.a

```

;; Glue for Printer Driver code resource headers MPW C.
;; header for each type of resource is placed in a separate
;; module using Asm PROC directive. The "-m" option to the
;; linker makes it the entry point to linked code resource.
;; Earle R. Horton. All rights reserved.
;; Tuesday, December 13, 1988
;; Some of Glue routines contain data. This is read-only!

CASE ON
SEG 'Main'
INCLUDE 'SysEqu.a'

myDrvFlags EQU ((1<<dCtlEnable) + (1<<dStatEnable) +
(1<<dNeedGoodBye)) << 8 ; Flag byte.
;; The header for the 'DRVr' resource.
DriverEntry PROC EXPORT ;; 'DRVr' starts here.
IMPORT myPrOpen ; open routine
IMPORT myPrPrime ; prime
IMPORT myPrControl ; control
IMPORT myPrStatus ; status
IMPORT myPrClose ; close
CODE

DC.W myDrvFlags ; control and status enable only
DC.W 0 ; doesn't need time
DC.W 0 ; no events
DC.W 0 ; no menu

DC.W myPrOpenCall-DriverEntry ; Offsets to driver
routines.
DC.W myPrPrimeCall-DriverEntry
DC.W myPrControlCall-DriverEntry
DC.W myPrStatusCall-DriverEntry
DC.W myPrCloseCall-DriverEntry
STRING PASCAL
DC.B '.Print' ; Driver name as Pascal string
DC.W 0 ; Zero for word-align.
dummyheader ; Make it look like a procedure for
LINK A6,#0 ; MacsBug
myPrOpenCall
PEA myPrOpen ;; SP -> desired driver routine.
BRA.S CallDriver ;; Jump interface to call real
routine.
myPrPrimeCall
PEA myPrPrime ;; Times 5.
BRA.S CallDriver
myPrControlCall
PEA myPrControl
BRA.S CallDriver
myPrStatusCall
PEA myPrStatus
BRA.S CallDriver
myPrCloseCall
PEA myPrClose
CallDriver
MOVE.L A1,-(SP) ;; Push ptr to dev control entry.
MOVE.L A0,-(SP) ;; Push pointer to parameter block.
MOVEA.L 8(SP),A0 ;; Get routine address.
JSR (A0) ;; Call driver routine.
;; Result is in D0.
MOVE.L (SP)+,A0 ;; Restore registers.
MOVE.L (SP)+,A1 ;; C programs leave stack alone.
ADDQ.W #$4,A7 ;; Fix up stack pointer.
BTST #$01,$0006(A0) ;; Immediate call?
BNE.S StdReturn ;; Yes, regular return.
MOVE.L JIODone,-(sp) ;; No, return via JIODone
StdReturn

```

```

RTS
label          ;; This is for MacsBug.
UNLK  A6
RTS
STRING ASIS
DC.B  'DMP-110 '
ENDP
;; The header for the Chooser Device Package.
CASE OFF

PackEntry PROC EXPORT
IMPORT device
BRA.S devicejump
deviceID
DC.W 0
Packname
DC.L $5041434B ; 'PACK'
IDnumber
DC.W $F000 ; -4096
Version
DC.W 1
Flags
DC.L $0C00E000
devicejump
JMP device
ENDP

;; The header for the Job and Style Dialogs code.
CASE OFF
DialogsEntry PROC EXPORT
import MyPrintDefault
import MyPrStlDialog
import MyPrJobDialog
import MyPrStlInit
import MyPrJobInit
import MyPrDlgMain
import MyPrValidate
import MyPrJobMerge
bra.w MyPrintDefault
bra.w MyPrStlDialog
bra.w MyPrJobDialog
bra.w MyPrStlInit
bra.w MyPrJobInit
bra.w MyPrDlgMain
bra.w MyPrValidate
bra.w MyPrJobMerge
ENDP

;; Routines to call Pascal procs which are known by address
;; and not by name. (ProcPtr glue).
CASE OFF
CallDlgInit PROC EXPORT
Export CallItemProc
CallItemProc
MOVE.L (SP)+,A0 ; Pop return address.
MOVE.L (SP)+,A1 ; Pop actual function address.
MOVE.L A0,-(SP) ; Push return address.
JMP (A1) ; Call the Pascal Function
ENDP

;; The header for the printing code.
CASE OFF ; If main code is in Pascal.

PrintEntry PROC EXPORT
IMPORT myPrOpenDoc
IMPORT myPrCloseDoc
IMPORT myPrOpenPage
IMPORT myPrClosePage
bra.w myPrOpenDoc
bra.w myPrCloseDoc
bra.w myPrOpenPage
bra.w myPrClosePage
ENDP
CASE ON
serialdata PROC EXPORT
; This module consists of read-only data.

```

```

baud300 EQU 380
baud600 EQU 189
baud1200 EQU 94
baud1800 EQU 62
baud2400 EQU 46
baud3600 EQU 30
baud4800 EQU 22
baud7200 EQU 14
baud9600 EQU 10
baud19200 EQU 4
baud57600 EQU 0

```

```

EXPORT AOutName,BOutName,BaudRates,prhires1f
; Handy storage place for constant strings & data which can
; be copied by the Driver routines.

```

```

STRING PASCAL
AOutName
DC.B '.AOut'
BOutName
DC.B '.BOut'
STRING ASIS
prhires1f
dc.b 3,26,27,'G'

```

```

ALIGN 2
BaudRates
DC.W baud300
DC.W baud600
DC.W baud1200
DC.W baud1800
DC.W baud2400
DC.W baud3600
DC.W baud4800
DC.W baud7200
DC.W baud9600
DC.W baud19200
DC.W baud57600
ENDP
END

```

Listing: Pack.c

```

/* Chooser Device Pack code. Described in Device Manager
 * chapter of Inside Macintosh IV. */
/* Earle R. Horton.
 * Wednesday, November 30, 1988
 * All rights reserved. */
#include <Resources.h>
#include <ToolUtils.h>
#include <OSUtils.h>
#include <Dialogs.h>
#include <Lists.h>
#define PACK
#include "dmp-110.h"
#include "compat.h"
#define __SEG__ Main
/* The Pascal Interface to MPW C 2.0.2 does not fully include
the List Manager. */
#ifdef macintosh
pascal void LGetCell(Ptr,short *,Cell,ListHandle);
pascal void LSetCell(Ptr,short,Cell,ListHandle);
pascal void LSetSelect(Boolean,Cell,ListHandle);
#endif

pascal OSErr device(message,caller,objname,zonename,p1,p2)
short message,caller;
StringPtr objname,zonename;
long p1,p2;
{
    Pfg settings;
#ifdef BACKWARD_COMPATIBLE
    SysEnvRec World;
    (void)SysEnvirons(1,&World);
    if(World.systemVersion < SYSNEEDEDX
        (void)StopAlert(SYSVERSALERT,nil);

```

```

    }
    else
#endif
    switch(message){
        case buttonMsg:
            prsetup(p2);
            break;
        case fillListMsg:
            filllist(p1);
            break;
        case selectMsg:
            settings = (Pfg)(GetResource('HEXA',RES2ID));
            pbaud = p2;
            break;
    }
    return (noErr);
}
/* Fill the list passed to us with the names of baud rates
 * contained in our 'STR#' resource. */
filllist(list)
ListHandle list;
{
    Pfg settings;
    Cell cell;
    short i;
    char thestring[256];
    settings = (Pfg)(GetResource('HEXA',RES2ID));
    (void)LAddRow(11,0,list);
    for(i=0;i<11;){
        SetPt(&cell,0,i++);
        GetIndString(thestring,RES1ID,i);
        LSetCell(&thestring[11],(short)thestring[0],cell,list);
    }
    SetPt(&cell,0,pbaud);
    LSetSelect(0xFFFF,cell,list);
    LAutoScroll(list);
}
/* prsetup() - Configure serial port.*/
prsetup(button)
long button;
{
    Pfg settings;
    if ((settings = (Pfg)(GetResource('HEXA',RES2ID))) == nil)
        return;
    if((button & 0xFF) == 1){
        if(IsMPPOpen()){ /* Conflict! */
            (void)StopAlert(ATALKALERT,nil);
            return;
        }
        pport = 1;
    }else{
        pport = 0;
    }
    ChangedResource(settings);
    WriteResource(settings);
}

```

Listing: pdef\$.c

```

/* Draft mode, text-only printing code for the Macintosh.*/
/* file is part of DMP-110 printer driver for the Macintosh
 * series of computers.*/
/* Earle R. Horton.
 * Wednesday, November 30, 1988
 * All rights reserved. */
#include <Windows.h>
#include <Events.h>
#include <Dialogs.h>
#include <Fonts.h>
#include <Memory.h>
#include <Resources.h>
#include <ToolUtils.h>
#include <Errors.h>
#include <OSUtils.h>
#include <Desk.h>
#include "dmp-110.h"
#include "compat.h"

```

```

#define __SEG__ Main

/* Use our own names for these.*/
#define storage 1GParam1
#define pagenum 1GParam2
#define PREC 1GParam3
#define picture 1GParam4
/* Useful constants */
#define SERRESET 8
#define SERSHAKE 10
#define XONCR ((char)17)
#define XOFFCR ((char)19)
#define RESFILEID (-8192)

OSErr SPOpen();

#define WNETrapNumber 0x60
#define UnImpTrapNumber 0x9f
/* Abort printing if command '.' pressed. */
checkabort()
{
    EventRecord myevent;
    Boolean result;
    Boolean WNEisThere = NGetTrapAddress(WNETrapNumber,1) !=
        NGetTrapAddress(UnImpTrapNumber,1);
    if(WNEisThere){
        result = WaitNextEvent(everyEvent,&myevent,0L,0L);
    }else{
        SystemTask();
        result = GetNextEvent(keyDownMask|autoKeyMask, &myevent);
    }
    if(result){
        if(LoWord(myevent.message & charCodeMask) == '.' &&
            (myevent.modifiers & cmdKey) ){
            PrSetError(iIOAbortErr); /* iPrAbort? */
        }
    }
}

/* This function returns a pointer to a specialized GrafPort
 * (a TPrPort) customized for printing.
 * Based on passed Print Record, prepare a GrafPort for use
 * by application in printing. store information of use to
 * us in 1GParam[n] fields of the TPrPort. We use a BitMap
 * whose baseAddr contains pointer to a buffer whose size is
 * taken from prXInfo field of print record. We define the
 * portRect to correspond to prInfo.rPage field of the print
 * record. Later, will scale application's drawing commands
 * to fit into prInfo.rPage Rect, and print Picture which
 * the application has created. */

pascal TPrPort myPrOpenDoc(hPrint,pPrPort,pIOBuf)
THPrint hPrint;
TPrPort pPrPort;
Ptr pIOBuf;
{
    TPrPort thisport;
    Handle us;
    THPrint savePrint;
    BitMap mybits;
    THPrint ourPrint;
    DCtlHandle d;
    DPStorage s;
    SysEnvRec World;
    short refnum;
    /* Allocate storage, abort if no space for TPrPort.
     * First stage aborts if not enough for a TPrPort, setting
     * PrError() to memFullErr and returning nil. */
    if(pPrPort == nil){
        if((thisport = (TPrPort)
            NewPtr((Size)sizeof(TPrPort))) == nil){
            PurgeMem(maxSize);
            thisport = (TPrPort)NewPtr((Size)sizeof(TPrPort));
            if(thisport == nil){
                PrSetError(memFullErr);
            }
        }
    }
}

```

```

        return nil;
    }
}
thisport->fOurPtr = TRUE; /* TRUE: printing code. */
} else {
    thisport = pPrPort;
    thisport->fOurPtr = FALSE; /* FALSE: application. */
}
thisport->fOurBits = TRUE;
/* If we get here, there is enough space for least TPrPort.
* next block of code attempts to allocate dynamic storage
* we will need during printing. If it fails, it sets
* thisport->storage to nil, & sets PrError() to memFullErr.
* application should detect the print error condition, and
* call PrCloseDoc. Dynamic storage which was not allocated
* successfully is disposed of by PrCloseDoc. Storage
* is allocated in several blocks rather than single big
* block in order to facilitate changing size of the output
* buffer or band. */
thisport->storage = (long)(s =
(DPstorage)NewPtr((Size)sizeof(Dstorage)));
if(s != nil){
    memset(s,0,sizeof(Dstorage));
    s->obuf = NewPtr((Size)2000);
}
ourPrint = hPrint;
if(HandToHand(&ourPrint) == noErr){
    thisport->PREC = (long)ourPrint;
} else {
    thisport->PREC = nil;
    thisport->gPort.portBits.baseAddr = mybits.baseAddr =
NewPtr((Size)
(((*hPrint)->prXInfo.iRowBytes*(*hPrint)->prXInfo.iBandV));
    if( (thisport->storage == nil) ||
        (s->obuf == nil) || (thisport->PREC == nil) ||
        (mybits.baseAddr == nil) ){
        PrSetError(memFullErr);
    } else {
/* If we get here, then memory worries are over, except of
* course if we run out while defining a pict. Don't know
* what happens then. TPrPort is ready to be initialized,
* and serial port to be opened. Only possible fatal error
* remaining is a serial port error.*/

        OpenPort(thisport);
        SetStdProcs(&thisport->gProcs); /* Fill out gProcs
for this port. */
        thisport->gPort.grafProcs = &thisport->gProcs;
        SetPort(thisport);
        SysEnviroms(1,&World);
        PrSetError(SPOpen(s,&World));

/* Save our copy of Print Record Handle, then open port. */
        thisport->PREC = (long)ourPrint;

/* Define the port's BitMap as our printing band.*/
        mybits.bounds.top = 0;
        mybits.bounds.left = 0;
        mybits.bounds.bottom = (*hPrint)->prXInfo.iBandV;
        mybits.bounds.right = (*hPrint)->prXInfo.iBandH+1;
        mybits.rowBytes = (*hPrint)->prXInfo.iRowBytes;
        SetPortBits(&mybits);

        MovePortTo((*hPrint)->prInfo.rPage.left,(*hPrint)-
>prInfo.rPage.top);
        PortSize((*hPrint)->prInfo.rPage.right-(*hPrint)-
>prInfo.rPage.left, (*hPrint)->prInfo.rPage.bottom-(*hPrint)-
>prInfo.rPage.top);

        RectRgn(thisport->gPort.visRgn,&thisport-
>gPort.portRect);
        RectRgn(thisport->gPort.clipRgn,&thisport-
>gPort.portRect);

```

```

        thisport->pagenum = 1;
        thisport->storage = (long)s;
        if((*ourPrint)->prJob.pIdleProc == nil)
            (*ourPrint)->prJob.pIdleProc =
(ProcPtr)checkabort;
/* Attempt to save Print Record in 'PREC' 1 in printer file.
* No big deal if failure, since it is not essential to
* printing. */
        savePrint = (THPrint)GetResource('PREC',1);
        if(savePrint != nil){
            LoadResource(savePrint);
            **savePrint = **ourPrint;
            ChangedResource(savePrint);
            UpdateResFile(HomeResFile(savePrint));
        }
    }
    return thisport;
}
/* Called at completion of print job, whether successful or
* not. Clean up any dynamic storage which requires it, then
* set print error to noErr. */
pascal void myPrCloseDoc(pPrPort)
TPPrPort pPrPort;
{
    DPstorage s;
    if( (pPrPort != nil) &&
        ((s = (DPstorage)pPrPort->storage) != nil) &&
        (s->obuf != nil) && (pPrPort->PREC != nil) &&
        (pPrPort->gPort.portBits.baseAddr != nil) ){
/* Assume successful open. */
        ClosePort(pPrPort);
        PBClose(&s->iopb,FALSE);
    }
    if(pPrPort != nil){ /* Dispose storage which needs it.*/
        if(s != nil){
            if(s->obuf != nil){
                DisposPtr(s->obuf);
            }
            DisposPtr(s);
        }
        if(pPrPort->PREC != nil){
            DisposHandle(pPrPort->PREC);
        }
        if(pPrPort->gPort.portBits.baseAddr != nil){
            DisposPtr(pPrPort->gPort.portBits.baseAddr);
        }
        if(pPrPort->fOurPtr){
            DisposPtr(pPrPort);
        }
        PrSetError(noErr); /* No bad feelings! */
    }
/* routine opens a new page. Check for valid TPrPort, then
* just start a new picture definition. */
pascal void myPrOpenPage(pPrPort,pPageFrame)
TPPrPort pPrPort;
TPRect pPageFrame;
{
    Rect *scalerect;
    if(pPageFrame != nil){
        scalerect = pPageFrame;
    } else {
        scalerect = &pPrPort->gPort.portRect;
    }
    SetPort(pPrPort);
/* app. may run out of memory while the picture is being
* saved. Don't know how to prevent or even detect this.*/
    pPrPort->picture = (long)OpenPicture(scalerect);
}
/* This is the routine which does the actual printing.
* Close picture for the page. Use DrawPicture() to draw
* repeatedly, while moving BitMap down the page to capture
* an image for each band.
* Send each band to the output conversion routines. */
pascal void myPrClosePage(pPrPort)
TPPrPort pPrPort;
{

```

```

PicHandle thepic;
short i, iocount, vband, nbands, rows, nleft;
Rect savepr, scaleRect, savebm;
long width, height;
THPrint ourPrint;
char formfeed;
DPStorage s;
ourPrint = (THPrint)pPrPort->PREC; /* Set up vars. */
SetPort(pPrPort);
ClosePicture();
thepic = (PicHandle)pPrPort->picture;
vband = (*ourPrint)->prXInfo.iBandV;
nbands = (*ourPrint)->prXInfo.iBands;
s = (DPStorage)pPrPort->storage;
if (PrError() != noErr) {
} else if (setjmp(s->abortbuf) != 0) { /* Pops back here
on abort or error. */
    PrSetError(iIOAbortErr);
} else if (pPrPort->pagenum < (*ourPrint)->prJob.iFstPage) {
/* Check page no. */
    pPrPort->pagenum++;
} else if (pPrPort->pagenum++ > (*ourPrint)-
>prJob.iLstPage) {
    PrSetError(iIOAbortErr);
} else {
    savepr = pPrPort->gPort.portRect;
    savebm = pPrPort->gPort.portBits.bounds;

/* Enlarge portRect of Printing Port. */
    OffsetRect(&pPrPort->gPort.portBits.bounds, -
savebm.left, -savebm.top);
    scaleRect = (*ourPrint)->prInfoPT.rPage;
    OffsetRect(&scaleRect, -scaleRect.left, -scaleRect.top);
    pPrPort->gPort.portRect = scaleRect;
    RectRgn(pPrPort->gPort.visRgn, &scaleRect);
    RectRgn(pPrPort->gPort.clipRgn, &scaleRect);

    rows = scaleRect.bottom - scaleRect.top;
    if ((*ourPrint)->prStl.feed != feedCut || wait-
nextpage()) {
        for (i=0; i<rows; i+=vband) { /* Loop over bands. */
            MovePortTo(scaleRect.left, -i); /* to next. */
            EraseRect(&scaleRect);
            DrawPicture(thepic, &scaleRect);
/* Draw in the band. */
            dumpbits(pPrPort); /* Print it. */
        }
        formfeed = 12; /* formfeed. Device Dependent. */
        s->iopb.ioParam.ioReqCount = 1;
        s->iopb.ioParam.ioBuffer = &formfeed;
        asyncwrite(s, *((THPrint)pPrPort->PREC))-
>prJob.iIdleProc);
/* Restore portRect so Application can draw in it. */
        pPrPort->gPort.portBits.bounds = savebm;
        pPrPort->gPort.portRect = savepr;
        RectRgn(pPrPort->gPort.visRgn, &savepr);
        RectRgn(pPrPort->gPort.clipRgn, &savepr);
    } else {
        PrSetError(iIOAbortErr); /* iPrAbort? */
    }
}
KillPicture(thepic);
dumpbits(pPrPort) /* Dump printing band to serial port. */
TPPrPort pPrPort;
{
    DPStorage s;
    int i, rows;
    BitMap *bits;
    ProcPtr idle;
    s = (DPStorage)pPrPort->storage;
    bits = &pPrPort->gPort.portBits;
    rows = bits->bounds.bottom-bits->bounds.top;

```

```

    idle = (*((THPrint)pPrPort->PREC))->prJob.iIdleProc;
    for (i=0; i<rows; i+=16) {
        bitmap_to_hires(s, bits, i, rows-i, idle);
        if (PrError() != noErr) return;
    }
}
/* bitmap_to_hires()
* function translates a QuickDraw BitMap to codes which may
* be sent to a Tandy DMP-110 dot-matrix printer in
* high-resolution graphics mode. Graphics codes for this
* printer in hi-res mode include all eight bit characters.
* There are two characters for each column of 16 dots on
* paper. The top dot (1) corresponds to bit
* zero of the first byte sent. The bottom dot (16)
* corresponds to bit eight of second byte. There are 960
* columns, 1920 bytes, of graphics data to be sent for one
* line of graphics output. Note that the ToolBox bit
* manipulation routines use lower-to-upper bit order.
* No smarts in routine, the whole 1918 bytes are sent for
* each line.*/
bitmap_to_hires(s, b, row, nleft, idle)
DPStorage s;
BitMap *b; /* BitMap to print. */
int row; /* Starting row on this pass. */
int nleft; /* Max number of rows to print. */
ProcPtr idle;
{
    short *obytes, *inbytes;
    int column, lastcol, nbits, dot;
    obytes = (short *)s->obuf;
    for (dot=960; dot>0; dot--) { obytes[dot] = 0; }
    lastcol = b->bounds.right - b->bounds.left;
    lastcol = (lastcol > 959) ? 959 : lastcol;
    inbytes = (short *) (b->baseAddr + b->rowBytes*row);
    nbits = b->rowBytes*8;
    for (column=0; column<lastcol; column++) {
        (*idle)();
        if (PrError() != noErr) return;
        for (dot=0; dot<8 && dot<nleft; dot++) {
            if (BitTst(inbytes, (long)(column+dot*nbits))) {
                BitSet(obytes, (long)(7-dot));
            }
        }
        for (dot=8; dot<16 && dot < nleft; dot++) {
            if (BitTst(inbytes, (long)(column+dot*nbits))) {
                BitSet(obytes, (long)(23-dot));
            }
        }
        obytes++;
    }
    output_hires_data(s, (long)(lastcol), idle);
/* Send to printer. */
}
/* output_hires_data() - Send a stream of high resolution
* graphics data to the Tandy DMP-110. The codes for
* transferring the graphics, and for high-resolution paper
* advance, are hard-coded into this routine. Skips blank
* graphics codes, and repositions the printer head. The
* overhead is 8 bytes for repositioning the head and
* restarting high resolution graphics. If there are four
* blank columns, then, we break even. If more, we
* win. If there are less four blank columns, we lose.*/
output_hires_data(s, columns, idle)
DPStorage s;
long columns;
ProcPtr idle;
{
    unsigned char codes[10];
    short *p;
    short pos, ncodes, start;
    short *buf = (short *)s->obuf;
    for (pos=0, ncodes=0, p=buf, start=0; pos<960; pos++) {
        if (buf[pos] == 0) {
            if (ncodes != 0) {
                sendcodes(s, p, ncodes, start, idle);
                if (PrError() != noErr) return;
            }

```

```

        ncodes = 0;
    }
} else {
    if(ncodes == 0){
        p = &buf[pos];
        start = pos;
    }
    ncodes++;
}
}
if(ncodes != 0){
    sendcodes(s,p,ncodes,start,idle);
}
codes[0] = 26;          /* DMP-110 carriage return. */
codes[1] = 27;          /* High res line feed. */
codes[2] = 'G';
s->iopb.ioParam.ioReqCount = 3L;
s->iopb.ioParam.ioBuffer = (Ptr)codes;
asyncwrite(s,idle);
}
sendcodes(s,buf,n,pos,idle)
DPStorage s;
char *buf;
int n,pos;
ProcPtr idle;
{
    unsigned char codes[10];
    codes[0] = 27;
    codes[1] = 16;
    codes[2] = (pos>>8)&3;
    codes[3] = pos & 0xFF;
    codes[4] = 27;
    codes[5] = 73;
    codes[6] = (n>>8)&3;
    codes[7] = n&0xFF;
    s->iopb.ioParam.ioReqCount = 8L;
    s->iopb.ioParam.ioBuffer = (Ptr)codes;
    asyncwrite(s,idle);
    s->iopb.ioParam.ioReqCount = (long)(2*n);
    s->iopb.ioParam.ioBuffer = buf;
    asyncwrite(s,idle);
}
/* Modal dialog box: "Insert next sheet." */
waitnextpage()
{
    DialogPtr sheetdialog;
    WindowPtr tempport;
    short itemhit,donetype;
    Handle doneitem;
    Rect donebox;
    if((sheetdialog = GetNewDialog(SHEETDIALOG, 0L,(WindowPtr)-1)) == nil)
        return FALSE;
    InitCursor();
    GetDItem(sheetdialog,DONEITEM,&donetype,&doneitem,&donebox);
    GetPort(&tempport);
    SetPort(sheetdialog);
    PenSize(3,3);
    InsetRect(&donebox,-4,-4);
    FrameRoundRect(&donebox,16,16);
    ModalDialog(0L,&itemhit);
    DisposDialog(sheetdialog);
    SetPort(tempport);
    if(itemhit == STOPITEM) return FALSE;
    return TRUE;
}
/* Open serial driver & configure it. Quit if ioResult field
 * of parameter block ever becomes other than noErr. */
OSErr SPOpen(s,World)
register DPStorage s;
SysEnvRec *World;
{
    register ParmBlkPtr pb;
    int serconfig;

```

```

    Pfg settings;
    OSErr error;
    extern char AOutName[],BOutName[];
    extern short BaudRates[];
    error = noErr;
    settings = (Pfg)(GetResource('HEXA',RESFILEID));
    if(settings == nil){ /* Where's our stuff? */
        return ResError();
    }
    LoadResource(settings);
    HNoPurge(settings);
    pb = &s->iopb;
    switch (pport){ /* get the correct port */
        case 0: /* modem port */
            pb->iopb.ioNamePtr = (StringPtr)AOutName;
            break;
        case 1: /* printer port */
            if(IsMPPOpen()){
                (void)StopAlert(ATALKALERT,nil);
                return portInUse;
            }
            pb->iopb.ioNamePtr = (StringPtr)BOutName;
            break;
    }
    PBOpen(pb,FALSE);
    if (pb->iopb.ioResult != noErr){
        return(pb->iopb.ioResult);
    }
    pb->iopb.ioNamePtr = nil;
    /* Set up io parameter block for writing to serial driver.
     * a control call resets the baud rate.
     * Another sets flow control. */
    ((CntrlParam *)pb)->csCode = SERRESET;
    serconfig = data8 + noParity + stop20;
    serconfig += BaudRates[pbaud];
    ((CntrlParam *)pb)->csParam[0] = serconfig;
    PBControl(pb,FALSE);
    if (pb->iopb.ioResult != noErr) return(pb->iopb.ioResult);
    #define shake ((SerShk *)&((CntrlParam *)pb)->csParam[0])
    shake->errs = FALSE;
    shake->evts = FALSE;
    shake->fDIR = FALSE;
    shake->fInX = FALSE;
    if(XonXoff && (World->machineType >= envMachUnknown)){
        shake->fXOn = TRUE;
        shake->fCTS = FALSE;
        shake->xOn = XONCR;
        shake->xOff = XOFFCR;
    } else {
        shake->fXOn = FALSE;
        shake->fCTS = TRUE;
    }
    ((CntrlParam *)pb)->csCode = SERSHAKE;
    PBControl(pb,FALSE);
    if (pb->iopb.ioResult != noErr) return(pb->iopb.ioResult);
    pb->iopb.ioPosMode = 0;
    pb->iopb.ioPosOffset = 0;
    ReleaseResource(settings);
    return(noErr);
}
/* Routine to write asynchronously to serial port, run an
 * idle proc.idle routine may call PrSetError(),
 * so we check PrError() for abort each time. Kill pending
 * serial port IO if abort detected.*/
asyncwrite(s,idle)
DPStorage s;
ProcPtr idle;
{
    IOParam *p = &(s->iopb.ioParam);
    PBWrite(p,TRUE); /* Issue ASYNC write. */
    do {
        (*idle)(); /* Idle til done. */
        if(PrError() != noErr){ /* Check for abort. */
            if(p->iopb.ioResult > 0){ /* More chars? */
                PBKillIO(p,FALSE); /* Stop output. */
            }
        }
    } while (p->iopb.ioResult == noErr);
}

```



```

    }
    longjmp(s->abortbuf, 1); /* Get out. */
}
}
while(p->ioreult > 0); /* Check for complete. */
if(p->ioreult < 0){ /* Serial port Error! */
    longjmp(s->abortbuf, 1);
}
}

Listing: pdef4.c
/* Code to implement Printing Dialogs. Described in rather
 * complete fashion in Technical Note #95. */
/* This file is part of the DMP-110 printer driver for the
 * Macintosh series of computers. */
/* Earle R. Horton.
 * Wednesday, November 30, 1988
 * All rights reserved. */
#include <Windows.h>
#include <Events.h>
#include <Dialogs.h>
#include <Controls.h>
#include <Lists.h>
#include <Fonts.h>
#include <Memory.h>
#include <Resources.h>
#include <ToolUtils.h>
#include <Packages.h>
#include <Menus.h>
#include "dmp-110.h"
#include "compat.h"
/* The Pascal Interface to MPW C 2.0.2 does not fully include
the List Manager. */
#ifdef macintosh
pascal void LGetCell(Ptr, short *, Cell, ListHandle);
pascal void LSetCell(Ptr, short, Cell, ListHandle);
pascal void LSetSelect(Boolean, Cell, ListHandle);
#endif

#ifdef iDevDaisy
#define bDevDaisy 2
#define iDevDaisy (bDevDaisy << 8)
#endif
#define STYLEDIALOG (0xE000)
#define JOBDIALOG (0xE001)
#define XTRA 24 /* Six extra longs for our use. */
#define NORMALPRINT 0
#define DLGSIZE ((long)(sizeof(TPrDlg) + XTRA))
#define extra(tp, i) (*((long *)((char *) (tp) +
sizeof(TPrDlg)) + (i)))
#define TPSIZE ((long)(sizeof(TPrint)))
/* Items we handle in the job and style dialogs. */
#define CANCELITEM 2
#define STYLETITLE 4
#define STYLELIST 5
#define ALLBUTTON 5
#define RANGEBUTTON 6
#define FROMNUM 7
#define TONUM 9
#define COPIES 11
#define FANBUTTON 13
#define SHEETBUTTON 14
#ifdef MPU68000 /* Aztec does not yet do prototypes. */
pascal void CallItemProc();
pascal TPrDlg CallDlgInit();
#else
pascal void CallItemProc(tp, itemhit, RealAddr) /* Glue. */
TPrDlg tp;
short itemhit;
ProcPtr RealAddr;
extern;
pascal TPrDlg CallDlgInit(hPrint, RealAddr) /* Glue. */
THPrint hPrint;
ProcPtr RealAddr;
extern;

```

```

#endif
/* Validate/update a print record. */
pascal Boolean MyPrValidate(hPrint)
THPrint hPrint;
{
    THPrint thedefault;
    if(Valid(hPrint)) return FALSE;
    else{
        thedefault = (THPrint)(GetResource('PREC', NORMALPRINT));
        LoadResource(thedefault);
        **hPrint = **thedefault;
        return TRUE;
    }
}
/* Printing dialog supervisor function.
 * Ref: Macintosh Technical Note 95. */
pascal Boolean MyPrDlgMain(hPrint, pDlgInit)
THPrint hPrint;
ProcPtr pDlgInit;
{
    TPrDlg tp;
    WindowPtr tempport;
    short donetype, itemhit;
    Handle doneitem;
    Rect donebox;
    ProcPtr itemproc;
    tp = CallDlgInit(hPrint, pDlgInit);
    itemproc = tp->pItemProc;
    tp->fDone = FALSE;
    tp->fDoIt = FALSE;
    GetPort(&tempport);
    SetPort(tp);
    ShowWindow(tp);
    GetDItem(tp, DONEITEM, &donetype, &doneitem, &donebox);
    PenSize(3, 3);
    InsetRect(&donebox, -4, -4);
    FrameRoundRect(&donebox, 16, 16);
    PenSize(1, 1);
    while(! (tp->fDone)){
        ModalDialog(tp->pFiltrProc, &itemhit);
        CallItemProc(tp, itemhit, itemproc);
        SetPort(tempport);
        CloseDialog(tp);
        DisposPtr(tp);
        DisposPtr(tp);
        if(tp->fDoIt) (void)MyPrValidate(hPrint);
        return(tp->fDoIt);
    }
}
/* Default item filter for print dialogs. */
pascal Boolean MyFilter(tp, theEvent, itemhit)
TPrDlg tp;
EventRecord *theEvent;
short *itemhit;
{
    char c;
    if(theEvent->what == keyDown){
        c = (theEvent->message & charCodeMask);
        if( (c == 13) || (c == 3) ){ /* Return or Enter. */
            *itemhit = 1;
            return TRUE;
        }
    }
    return FALSE;
}
/* Style Dialog Filter. Give it to List Manager if it is
 * in list of 'PREC's. Signal done if a double-click in a
 * cell. Otherwise, handle via the normal filter. */
pascal Boolean MyStlFilter(tp, theEvent, itemhit)
TPrDlg tp;
EventRecord *theEvent;
short *itemhit;
{
    if(*itemhit == STYLELIST){
        GlobalToLocal(&theEvent->where);
        (void)LCClick(theEvent->where, theEvent-

```

```

>modifiers,(ListHandle)extra(tp,0));
    LocalToGlobal(&theEvent->where);
    return FALSE;
} else {
    return MyFilter(tp,theEvent,itemhit);
}
}
/* This function fills a print record with defaults. The
 * default values are stored in Printer resource file, in
 * PREC 0. This is easy. */
pascal void MyPrintDefault(hPrint)
THPrint hPrint;
{
    THPrint thedefault;
    thedefault = (THPrint)(GetResource('PREC',NORMALPRINT));
    LoadResource(thedefault);
    **hPrint = **thedefault; /* What the hey. */
}
/* The next routine handles the style dialog. Two
 * possibilities exist. If the cancel button is hit, then we
 * signal quit. The print record is not changed.
 * If "OK" button is hit, then we have to fill in the new
 * style in the user print record. */
pascal void HandleStyleItems(tp,itemhit)
TPPrDlg tp;
short itemhit;
{
    short thenum,type;
    Handle item;
    Rect box,bigbox;
    MenuHandle PRECMenu;
    long restype;
    short resid;
    short nitems,i;
    THPrint rPrint;
    char thestring[256];
    Point theCell;
    switch(itemhit){
        case DONEITEM:
            SetPt(&theCell,0,0);
            LGetSelect(0xFFFF,&theCell,(ListHandle)extra(tp,0));
            i = 255;
            LGetCell(&thestring[1],&i,theCell,(ListHandle)extra(tp,0));
            thestring[0] = i;
            rPrint =
            (THPrint)GetNamedResource('PREC',thestring);
            if(rPrint == nil){
                rPrint = (THPrint)GetResource('PREC',0);
            }
            LoadResource(rPrint);
            (*(tp->hPrintUsr)->prInfo = (*rPrint)->prInfo;
            (*(tp->hPrintUsr)->prInfoPT = (*rPrint)->prInfoPT;
            (*(tp->hPrintUsr)->rPaper = (*rPrint)->rPaper;
            (*(tp->hPrintUsr)->prStl = (*rPrint)->prStl;
            (*(tp->hPrintUsr)->prXInfo = (*rPrint)->prXInfo;
            (*(tp->hPrintUsr)->printX[0] = (*rPrint)->printX[0];
            LDispose((ListHandle)extra(tp,0));
            tp->fDone = TRUE;
            tp->fDoIt = TRUE;
            break;
        case CANCELITEM:
            LDispose((ListHandle)extra(tp,0));
            tp->fDone = TRUE;
            tp->fDoIt = FALSE;
            break;
        default:
            break;
    }
}
/* Symbolic constants used to make LNew() more readable. */
#define Drawn 0xFFFF
#define noGrow 0x0000
#define noHScroll 0x0000

```

```

#define vScroll 0xFFFF
#include <strings.h>
/* User item for putting up list of available Print Recs. */
pascal void PRECUserItem(tp,theitem)
TPPrDlg tp;
short theitem;
{
    short type,i,j,nitems,id;
    Handle item;
    Rect listBox,dataBounds;
    Point cellSize,theCell;
    char thestring[256];
    THPrint rPrint;
    long restype;
    ListHandle PRECList;
    /* First time called, create the list. */
    if(extra(tp,0) == nil){
        GetDItem(tp,theitem,&type,&item,&listbox);
        listBox.right -=15;
        SetRect(&dataBounds,0,0,1,0);
        SetPt(&cellSize,0,0);
        PRECList =
        LNew(&listbox,&dataBounds,cellSize,0,(WindowPtr)tp,
            Drawn,noGrow,noHScroll,vScroll);
        (*PRECList)->selFlags = 1DoHAutoScroll + 1OnlyOne;
        extra(tp,0) = (long)PRECList;
    }
    /* Read in the resources, put the names in the list. */
    nitems = CountResources('PREC');
    (void)LAddRow(nitems,0,PRECList);
    for(i=0,j=0;i++<nitems;){
        rPrint = (THPrint)GetIndResource('PREC',i);
        GetResInfo(rPrint,&id,&restype,thestring);
        SetPt(&theCell,0,j++);
        LSetCell(&thestring[1],(short)thestring[0],theCell,PRECList);
        if(((tp->hPrintUsr)->printX[0] == id){
            LSetSelect(0xFFFF,theCell,PRECList);
            LAutoScroll(PRECList);
        }
    }
}
/* After the first time, just frame item's Rect, and call
 * LUpdate() to draw the list. */
GetDItem(tp,theitem,&type,&item,&listbox);
InsetRect(&listbox,-1,-1);
FrameRect(&listbox);
LUpdate(tp->Dlg.window.port.visRgn,(ListHandle)extra(tp,0));
}
/* The style dialog initializer. Get the dialog from the
 * resource file, calculate Rect for Print Record list. */
pascal TPPrDlg MyPrStlInit(hPrint)
THPrint hPrint;
{
    TPPrDlg tp;
    short type;
    Handle item;
    Rect box,listbox;
    tp = (TPPrDlg)NewPtr(DLGSize);
    (GetNewDialog(STYLEIALOG,tp,(WindowPtr)-1));
    GetDItem(tp,STYLETITLE,&type,&item,&box);
    listBox.left = box.right+20;
    listBox.top = tp->Dlg.window.port.portRect.top+10;
    listBox.bottom = tp->Dlg.window.port.portRect.bottom-10;
    listBox.right = listBox.left + 200;
    GetDItem(tp,STYLELIST,&type,&item,&box);
    SetDItem(tp,STYLELIST,type,PRECUserItem,&listbox);
    extra(tp,0) = nil;
    tp->pFltrProc = (ProcPtr)MyStlFilter;
    tp->pItemProc = (ProcPtr)HandleStyleItems;
    tp->hPrintUsr = hPrint;
    return(tp);
}
pascal Boolean MyPrStlDialog(hPrint) /* Conduct printer
style dialog. */

```

```

THPrint hPrint;
{
    return(MyPrDlgMain(hPrint,MyPrStlInit));
}
/* Record user choice of paper feed, page numbers, and
 * number of copies. */
pascal void HandleJobItems(tp,itemhit)
TPPrDlg tp;
short    itemhit;
{
    short    thenum;
    long     thelong;
    Handle   numitem;
    Rect     numbox;
    char     title[256];
    switch(itemhit){
        case DONEITEM:
            tp->fDone = TRUE;
            tp->fDoIt = TRUE;

            if(buttonset(tp,ALLBUTTON)){
                (*(tp->hPrintUsr))->prJob.iFstPage = iPrPgFst;
                (*(tp->hPrintUsr))->prJob.iLstPage = iPrPgMax;
            }else{
                GetDItem(tp,FROMNUM,&thenum,&numitem,&numbox);
                GetIText(numitem,&title[0]);
                StringToNum(&title[0],&thelong);
                (*(tp->hPrintUsr))->prJob.iFstPage = thelong;
                GetDItem(tp,TONUM,&thenum,&numitem,&numbox);
                GetIText(numitem,&title[0]);
                StringToNum(&title[0],&thelong);
                (*(tp->hPrintUsr))->prJob.iLstPage = thelong;
            }
            GetDItem(tp,COPIES,&thenum,&numitem,&numbox);
            GetIText(numitem,&title[0]);
            StringToNum(&title[0],&thelong);
            (*(tp->hPrintUsr))->prJob.iCopies = (short)thelong;
            if(buttonset(tp,SHEETBUTTON){
                (*(tp->hPrintUsr))->prStl.feed = feedCut;
            }
            else (*(tp->hPrintUsr))->prStl.feed = feedFanfold;
            break;
        case CANCELITEM:
            tp->fDone = TRUE;
            break;
        case SHEETBUTTON:
            pushradiobutton(tp,SHEETBUTTON,FANBUTTON,SHEETBUTTON);
            (*(tp->hPrintUsr))->prStl.feed = feedCut;
            break;
        case FANBUTTON:
            pushradiobutton(tp,FANBUTTON,FANBUTTON,SHEETBUTTON);
            (*(tp->hPrintUsr))->prStl.feed = feedFanfold;
            break;
        case ALLBUTTON:
            pushradiobutton(tp,ALLBUTTON,ALLBUTTON,RANGEBUTTON);
            break;
        case RANGEBUTTON:
            pushradiobutton(tp,RANGEBUTTON,ALLBUTTON,RANGEBUTTON);
            break;
        default:
            break;
    }
}
/* The job dialog initializer. Make sure all the buttons
 * come up reflecting the Print Record contents. */
pascal TPPrDlg MyPrJobInit(hPrint)
THPrint hPrint;
{
    TPPrDlg    tp;
    short      thenum;
    Handle      theitem;
    Rect        thebox;
    char        title[255];
    tp = (TPPrDlg)NewPtr(DLGSIZE);

```

```

        (GetNewDialog(JOBDIALOG,tp,(WindowPtr)-1));
        pushradiobutton(tp,ALLBUTTON,ALLBUTTON,RANGEBUTTON);
        if((*(hPrint))->prStl.feed == feedCut)
            pushradiobutton(tp,SHEETBUTTON,FANBUTTON,SHEETBUTTON);
        else
            pushradiobutton(tp,FANBUTTON,FANBUTTON,SHEETBUTTON);
        GetDItem(tp,FROMNUM,&thenum,&theitem,&thebox);
        thenum = (*(hPrint))->prJob.iFstPage;
        NumToString((long)thenum,title);
        SetIText(theitem,title);
        GetDItem(tp,TONUM,&thenum,&theitem,&thebox);
        thenum = (*(hPrint))->prJob.iLstPage;
        NumToString((long)thenum,title);
        SetIText(theitem,title);
        GetDItem(tp,COPIES,&thenum,&theitem,&thebox);
        thenum = (*(hPrint))->prJob.iCopies /*= 1*/;
        NumToString((long)thenum,title);
        SetIText(theitem,title);
        tp->fFltrProc = (ProcPtr)MyFilter;
        tp->pItemProc = (ProcPtr)HandleJobItems;
        tp->hPrintUsr = hPrint;
        return(tp);
}
pascal Boolean MyPrJobDialog(hPrint) /* Conduct printer job
dialog. */
THPrint hPrint;
{
    return(MyPrDlgMain(hPrint,MyPrJobInit));
}
/* Copy a job subrecord. Update the destination record's
 * printer information, band information, and paper rect,
 * based on information in the job subrecord. */
pascal void MyPrJobMerge(hPrintSrc,hPrintDst)
THPrint hPrintSrc,hPrintDst;
{
    (*(hPrintDst))->prInfo.iDev = (*(hPrintSrc))->prInfo.iDev;
    (*(hPrintDst))->prJob = (*(hPrintSrc))->prJob;
    (*(hPrintDst))->prXInfo = (*(hPrintSrc))->prXInfo;
    (*(hPrintDst))->prPaper = (*(hPrintSrc))->prPaper;
    (*(hPrintDst))->prInfo = (*(hPrintSrc))->prInfo;
    (*(hPrintDst))->prInfoPT = (*(hPrintSrc))->prInfoPT;
}
/* Handy test for radio button down. */
int buttonset(d,num)
DialogPtr d;
short num;
{
    short type;
    Handle item;
    Rect box;
    GetDItem(d,num,&type,&item,&box);
    if(GetCtlValue(item) == 1){
        return true;
    }else{
        return false;
    }
}
pushradiobutton(thedialog,itemhit,first,last)
DialogPtr thedialog; /* set itemhit, unset */
int itemhit,first,last; /* all others in range */
{
    int itemtype,i;
    Handle itemhandle; /* Does check boxes, too. */
    Rect itemrect;
    if(first == 0) return;
    for(i=first-1;last-i++;){
        GetDItem(thedialog,i,&itemtype,&itemhandle,&itemrect);
        if(i == itemhit) SetCtlValue(itemhandle,1);
        else SetCtlValue(itemhandle,0);
    }
}
/* This function answers the question: Can we possibly use
 * this print record? Try to be liberal here. Don't
 * allow a Print Record which requires a larger buffer than

```

```

    * our printing routines can handle. */
int ValidChPrint)
THPrint hPrint;
{
    if (((*hPrint)->iPrVersion != VERSION) || ((*hPrint)-
>prInfo.iDev != 0) || ((*hPrint)->prJob.bJDocLoop !=
bDraftLoop))
        return FALSE;
    if ((*hPrint)->prXInfo.iBandV != 64 ||
        (*hPrint)->prXInfo.iBandH != 959 ||
        (*hPrint)->prXInfo.iRowBytes != 120
        )
        return FALSE;
    return TRUE;
}

```

Listing: PDriver.C

```

/* This is the driver. A description of the driver can be
 * found in Device Manager chapter of Inside Macintosh, & a
 * description of THIS driver can be found in Print Manager
 * chapter.*/
/* file is part of the DMP-110 printer driver for the Macin-
tosh series of computers.*/
/* Earle R. Horton.
 * Wednesday, November 30, 1988
 * All rights reserved.*/
#define DRIVER
#include <Windows.h>
#include <Events.h>
#include <Dialogs.h>
#include <Fonts.h>
#include <Memory.h>
#include <Resources.h>
#include <ToolUtils.h>
#include <Errors.h>
#include <Desk.h>
#include "dmp-110.h"
#include "compat.h"
#define __SEG__ Main
#define iPrEvtAll 0x00FFFFFD
#define iPrEvtTop 0x00FEFFFF
#define dNeedGoodBye 4
int checkabort();
/* Useful constants */
#define SERRESET 8
#define SERSHAKE 10
#define XONCR ((char)17)
#define XOFFCR ((char)19)
#define RESFILEID (-8192)

extern char AOutName[], BOutName[];
extern short BaudRates[];
extern char prlfstr[];
extern char printstr[];
extern char prtopstr[];
extern char preopstr[];
extern char prmargin[];
extern char phiresif[];

OSErr openprinterfile(), SP0pen();
/* Driver close routine.
 * Close serial port. Dispose of our storage.
 * Better not close the serial driver if we
 * are dealing with 64k ROMs.
 * Called by device manager under UniFinder when application
 * heap is reinitialized.*/
OSErr myPrClose(p,d) /* Device Close Call */
    PrParam *p; /* ==> parameter block */
    DCtlPtr d; /* ==> device control entry */
{
    return(noErr);
}
#include <strings.h>

```

```

/* Printer driver open routine. Open our printer resource
 * file, get any information we have stored, allocate a
 * non-relocatable block of storage, set up serial port for
 * use. Check errors.*/
OSErr myPrOpen(p,d)
    PrParam *p; /* ==> parameter block */
    DCtlPtr d; /* ==> device control entry */
{
    extern short DriverEntry;
    return (noErr);
}
OSErr myPrPrime(p,d) /* We don't prime. It's for drivers
which do */
    /* read/write directly. */
    PrParam *p; /* ==> parameter block */
    DCtlPtr d; /* ==> device control entry */
{
    return(noErr);
}
OSErr myPrControl(p,d) /* Control calls. Many defined, few
implemented.*/
    PrParam *p; /* ==> parameter block */
    DCtlPtr d; /* ==> device control entry */
{
    Dstorage storage;
    char *buf = nil;
    SysEnvRec World;
    OSErr error;
    ProcPtr idle;
    SysEnviron(1,&World);
    PrSetError(noErr);
    memset(&storage,0,sizeof(Dstorage));
    error = SP0pen(&storage,&World);
    if(error != noErr) return noErr;
    storage.iopb.ioParam.ioResult = noErr;
    if(setjmp(storage.abortbuf) != 0){
        PBClose(&storage.iopb);
        if(p->csCode == iPrEvtCtl){
            DisposPtr(buf);
        }
        return noErr;
    }
    /*Device Control Call. p->csCode gives opcode, and we switch
on it to perform low-level Printing calls*/
    switch (p->csCode){
        case iPrBitsCtl: /* Send a bitmap to the printer. */
            break;
        case iPrEvtCtl: /* Screen prntng. (cmd-shift 4.)*/
            if(*(short*)&p->lParam1) == 1 /* Top window.*/
                buf = NewPtr(2000L);
                if(buf != nil){
                    dumptop(&storage,buf);
                    DisposPtr(buf);
                }
            }
            else if(*(short*)&p->lParam1) == 2{
                /* Screen. */
                buf = NewPtr(2000L);
                if(buf != nil){
                    dumpscreen(&storage,buf);
                    DisposPtr(buf);
                }
            }
            break;
        default:
            break;
    }
    PBClose(&storage.iopb);
    return noErr;
}
/* Printer driver status call, used by the Font Manager to
 * request a copy of printer's font characterization table.
 * Ignored like the control call.*/

```

```

OSError myPrStatus(p,d) /* Device Status Call */
PrParam *p; /* ==> parameter block */
DcTIPtr d; /* ==> device control entry */
{
    return (noErr);
}
/* Open serial driver and configure it. Quit if ioResult
 * field of param block ever becomes other than noErr. */
OSError SPOpen(s,World)
register DPStorage s;
SysEnvRec *World;
{
    register ParmBlkPtr pb;
    int serconfig;
    Pfg settings;
    Handle pfilename;
    short pfile;
    pfilename = GetResource('STR',0xE000);
    if(pfilename == nil){
        return ResError();
    }
    HLock(pfilename);
    pfile = OpenResFile(*pfilename);
    if(pfile == -1){
        return ResError();
    }
    DisposHandle(pfilename);
    settings = (Pfg)(GetResource('HEXA',RESFILEID));
    if(settings == nil){
        return ResError();
    }
    LoadResource(settings);
    HNoPurge(settings);
    pb = &s->iopb;
    switch (pport){ /* get the correct port */
        case 0: /* modem port */
            pb->iuParam.ioNamePtr = (StringPtr)A0OutName;
            break;
        case 1: /* printer port */
            break;
        default:
            if(IsMPP0Open()){
                (void)StopAlert(ATAKALERT,nil);
                return portInUse;
            }
            pb->iuParam.ioNamePtr = (StringPtr)B0OutName;
            break;
    }
    PBOpen(pb,FALSE);
    if (pb->iuParam.ioResult != noErr){
        return(pb->iuParam.ioResult);
    }
    pb->iuParam.ioNamePtr = nil;
    /* Set up the io parameter block for writing to the serial
     * driver. a control call resets the baud rate */
    ((CntrlParam *)pb)->csCode = SERRESET;
    serconfig = data8 + noParity + stop20;
    serconfig += BaudRates[pbaud];
    ((CntrlParam *)pb)->csParam[0] = serconfig;
    PBControl(pb,FALSE);
    if (pb->iuParam.ioResult != noErr) return(pb->iuParam.ioResult);
    #define shake ((SerShk *)((CntrlParam *)pb)->csParam[0])
    shake->errs = FALSE;
    shake->evts = FALSE;
    shake->fDTR = FALSE;
    shake->fInX = FALSE;
    if((XonXoff && (World->machineType >= envMachUnknown)){
        shake->fXOn = TRUE;
        shake->fCTS = FALSE;
        shake->xOn = XONCR;
        shake->xOff = XOFCR;
    } else {
        shake->fXOn = FALSE;
        shake->fCTS = TRUE;
    }
    ((CntrlParam *)pb)->csCode = SERSHAKE;
    PBControl(pb,FALSE);
    if (pb->iuParam.ioResult != noErr) return(pb->iuParam.ioResult);
}

```

```

pb->iuParam.ioPosMode = 0;
pb->iuParam.ioPosOffset = 0;
return(noErr);
}
printstring(s,string) /* Send printer control string to */
/* serial driver. */
register DPStorage s;
unsigned char *string;
{
    s->iopb.iuParam.ioBuffer = (Ptr)(string+1);
    s->iopb.iuParam.ioReqCount = (long)(string[0]);
    asyncwrite(s,checkabort);
}
/* dumptop() - Called to dump the top window to the screen.
 * Nothing if top window is a color window.
 * Uses PtInRgn() to determine the extent of the window's
 * structure region, so we can print entire window frame.
 * If you know of a better way, let me know. Question:
 * What do we do if we get a round window? */
dumptop(s,obuf)
DPStorage s;
short *obuf;
{
    EventRecord myevent;
    GrafPtr tmpport,oldport;
    int i,rows,width,first_row,last_row,left_offset;
    Rect printrect;
    GetPort(&oldport);
    tmpport = (GrafPtr)FrontWindow();
    SetPort(tmpport);
    /* Expand portRect to Rect enclosing Window struct region. */
    printrect = tmpport->portRect;
    LocalToGlobal(&printrect);
    LocalToGlobal(&printrect.bottom);
    while(PtInRgn((Point*)&printrect),((WindowPeek)tmpport)->strucRgn){ printrect.left--; }
    printrect.left++;
    while(PtInRgn((Point*)&printrect),((WindowPeek)tmpport)->strucRgn){ printrect.top--; }
    printrect.top++;
    while(PtInRgn((Point*)&printrect.bottom),((WindowPeek)tmpport)->strucRgn){ printrect.bottom++; }
    printrect.bottom--;
    while(PtInRgn((Point*)&printrect.bottom),((WindowPeek)tmpport)->strucRgn){ printrect.right++; }
    printrect.right--;
    GlobalToLocal(&printrect);
    GlobalToLocal(&printrect.bottom);
    /* Calculate rows in BitMap which we need to print. Send
     * sixteen at a time to the BitMap printing routine.*/
    first_row = printrect.top - tmpport->portBits.bounds.top;
    last_row = printrect.bottom - tmpport->portBits.bounds.top;
    left_offset = printrect.left - tmpport->portBits.bounds.left;
    width = printrect.right - printrect.left;
    for(i=first_row;i<=last_row;i+=16){
        bitmap_to_hires(s,&tmpport->portBits,i,
            obuf,width,left_offset,last_row-i+1);
    }
    SetPort(oldport);
}
/* dumpscreen() - Called to dump the screen to the printer.
 * Dumps WMgrPort instead. Funky stuff if screen is not in
 * two-color mode.*/
dumpscreen(s,obuf)
DPStorage s;
short *obuf;
{
    GrafPtr tmpport;
    int i,rows,width;
    GetWMgrPort(&tmpport);
    rows = tmpport->portBits.bounds.bottom - tmpport->portBits.bounds.top;
    width = tmpport->portRect.right - tmpport->portRect.left - 1;
}

```

```

    for(i=0;i<rows;i+=16){
        bitmap_to_hires(s,&tmpport-
    }portBits,i,obuf,width,0,rows-i);
    }
}
/* bitmap_to_hires()
 * function translates a QuickDraw BitMap to codes which may
 * be sent to a Tandy DMP-110 dot-matrix printer in
 * high-resolution graphics mode. Graphics codes for this
 * printer in hi-res mode include all eight bit characters.
 * There are two characters for each column of 16 dots on
 * paper. The top dot (1) corresponds to bit
 * zero of the first byte sent. The bottom dot (16)
 * corresponds to bit eight of second byte. There are 960
 * columns, 1920 bytes, of graphics data to be sent for one
 * line of graphics output. Note that the ToolBox bit
 * manipulation routines use lower-to-upper bit order.
 * No smarts in this routine, whole 1918 bytes are sent for
 * each line. */
bitmap_to_hires(s,b,row,obuf,width,left_offset,nleft)
DPStorage s; /* Global driver storage. */
BitMap *b; /* BitMap to print. */
int row; /* Starting row on this pass. */
short *obuf; /* Serial port output buffer. */
int width; /* Print this many columns. */
int left_offset; /* Start this far from left of b-
>bounds.left. */
int nleft; /* Max number of rows to print. */
{
    short *obytes,*inbytes;
    int column,lastcol,nbits,dot;
    for(dot=960;dot->0;){
        obuf[dot] = 0;
    }
    obytes = obuf;
    lastcol = (b->bounds.right > 959) ? 959 : b->bounds.right;
    lastcol = (lastcol > width) ? width : lastcol;
    inbytes = (short *) (b->baseAddr + b->rowBytes*row);
    nbits = b->rowBytes*8;
    /* Transform QuickDraw BitMap to Tandy DMP-110 high resolution
    graphics using ToolBox bit-manipulation routines. Sixteen
    rows BitMap column become two-byte printing code.*/
    for(column=left_offset;column<=lastcol+left_offset;column++){
        for(dot=0;dot<8 && dot<nleft;dot++){
            if(BitTst(inbytes,(long)(column+dot*nbits))){
                BitSet(obytes,(long)(7-dot));
            }
        }
        for(dot=8;dot<16 && dot < nleft;dot++){
            if(BitTst(inbytes,(long)(column+dot*nbits))){
                BitSet(obytes,(long)(23-dot));
            }
        }
        obytes++;
    }
    output_hires_data(s,obuf,(long)(lastcol)-1); /* Send to
    printer. */
}
/* output_hires_data() - Send a stream of high resolution
 * graphics data to the Tandy DMP-110. The codes for
 * transferring the graphics, and for high-resolution paper
 * advance, are hard-coded into this routine. Skips blank
 * graphics codes, and repositions the printer head. The
 * overhead is 8 bytes for repositioning the head and
 * restarting high resolution graphics. If there are four
 * blank columns, then, break even. If there are more, we
 * win. If there are less than four blank columns, lose. */
output_hires_data(s,buf,columns)
DPStorage s;
short *buf;
long columns;
{
    short *p;
    short pos,ncodes,start,max;
    max = (columns > 959) ? 959 : columns;
    for(pos=0,ncodes=0,p=buf,start=0;pos<columns;pos++){

```

```

        if(buf[pos] == 0){
            if(ncodes != 0){
                sendcodes(s,p,ncodes,start);
                ncodes = 0;
            }
        }
        else{
            if(ncodes == 0){
                p = &buf[pos];
                start = pos;
                ncodes++;
            }
        }
        if(ncodes != 0){
            sendcodes(s,p,ncodes,start);
            printstring(s,prhires1f);
        }
    }
    sendcodes(s,buf,n,pos)
DPStorage s;
char *buf;
int n,pos;
{
    unsigned char codes[10];
    s->iopb.ioParam.ioBuffer = (Ptr)&codes[0];
    codes[0] = 27;
    codes[1] = 16;
    codes[2] = (pos>>8)&3;
    codes[3] = pos & 0xFF;
    codes[4] = 27;
    codes[5] = 73;
    codes[6] = (n>>8)&3;
    codes[7] = n&0xFF;
    s->iopb.ioParam.ioReqCount = 8;
    asyncwrite(s,checkabort); /* Send 8 bytes to printer. */
    s->iopb.ioParam.ioBuffer = &buf[0];
    s->iopb.ioParam.ioReqCount = 2*n;
    asyncwrite(s,checkabort); /* Print the buffer. */
}

/* checkabort() - Returns TRUE if command-'. ' detected.
 * Otherwise, FALSE. */
checkabort()
{
    EventRecord myevent;
    if(WaitNextEvent(everyEvent,&myevent,0L/*WaitTime*/,0L)){
        if(LoWord(myevent.message & charCodeMask) == '.' &&
            (myevent.modifiers & cmdKey) ){
            PrSetError(iIOAbortErr);
            return TRUE;
        }
    }
    return FALSE;
}

/* Routine to write asynchronously to serial port, and run
 * an idle proc while we wait. The idle routine may call
 * PrSetError(), so we check PrError() for abort each time.
 * Kill pending serial port IO if abort detected. */
asyncwrite(s,idle)
DPStorage s;
ProcPtr idle;
{
    IOParam *p = &(s->iopb.ioParam);
    PBWrite(p,TRUE); /* Issue ASYNC write. */
    do{
        (*idle)(); /* Idle til done. */
        if(PrError() != noErr){ /* Check for abort. */
            if(p->ioResult > 0){ /* More chars? */
                PBKillIO(p,FALSE); /* Stop output. */
            }
            longjmp(s->abortbuf,1); /* Get out. */
        }
    }while(p->ioResult > 0); /* Check for complete. */
}

```

Listing: DMP-110.rsrc.r
 /* contents of a Macintosh Printer Resource File can be found

in the Print Manager Chapter of Inside Macintosh, if you can get it. */

/* This file is part of DMP-110 printer driver for the Mac
* series of computers. */

#include "Types.r"

/* This file's creator. */

#define Version "DMP-110 v1.0"

```
type 'Dmp1' (
    pstring; /* String */
);
/* Print record resource file copy. */
type 'PREC' (
    integer; /* version */
    /* prInfo */
    byte;
    byte; /* iDev */
    integer; /* iVres */
    integer; /* iHres */
    rect; /* rPage */
    rect; /* rPaper */
    /* prStl */
    integer; /* wDev */
    integer; /* iPageV */
    integer; /* iPageH */
    byte; /* bPort */
    byte; /* feed */
    /* prInfoPT */
    byte;
    byte; /* iDev */
    integer; /* iVres */
    integer; /* iHres */
    rect; /* rPage */
    /* prXInfo */
    integer; /* iRowBytes */
    integer; /* iBandV */
    integer; /* iBandH */
    integer; /* iDevBytes */
    integer; /* iBands */
    byte; /* bPatScale */
    byte; /* bULThick */
    byte; /* bULOffset */
    byte; /* bULShadow */
    byte; /* scan */
    byte; /* bXInfoX */
    /* prJob */
    integer; /* iFstPage */
    integer; /* iLstPage */
    integer; /* iCopies */
    byte; /* bJDocLoop */
    byte; /* fFromUsr */
    longint; /* pIdleProc */
    longint; /* pFileName */
    integer; /* iFileVol */
    byte; /* bFileVers */
    byte; /* bJobX */
    array [19](
        integer; /* "Private" */
    );
);
```

```
resource 'ICN#' (128, "DMP-110") (
    /* array: 2 elements */
    /* [1] */
    "$0000 0000 0000 0000 0000 0000 0000 0000"
    "$0000 0000 0000 0000 0000 0000 007F 8000"
    "$0040 C000 0040 A000 0040 9000 0040 F800"
    "$0040 0800 0040 0800 0040 0800 0040 0800"
    "$0040 0800 0040 0800 03FF FFC0 04AA AC20"
    "$04D5 55A0 04AA ADA0 027F F840 0200 0040"
    "$01FF FF80 0008 0400 0008 0400 0008 0400"
    "$1E08 0E00 1FF0 1F00 1E00 1F00 0000 0A",
    /* [2] */
    "$0000 0000 0000 0000 0000 0000 0000 0000"
    "$0000 0000 0000 0000 0000 0000 007F 8000"
    "$007F C000 007F E000 007F F000 007F F800"
```

```
$"007F F800 007F F800 007F F800 007F F800"
"$007F F800 007F F800 03FF FFC0 07FF FFE0"
"$07FF FFE0 07FF FFE0 03FF FFC0 03FF FFC0"
"$01FF FF80 0008 0400 0008 0400 0008 0400"
"$1E08 0E00 1FF0 1F00 1E00 1F00 0000 0A"
```

```
);
resource 'FREF' (128) (
    'PRER',
    0,
    ""
);
resource 'Dmp1' (0) (
    Version
);
resource 'BNDL' (128) (
    'Dmp1',
    (
        'ICN#',
        (
            0, 128
        ),
        'FREF',
        (
            0, 128
        )
    )
);
data 'HEXA' (-8192, "Printer Settings") (
    "$0001 0002 0000 "
);
resource 'STR' (-4092, "Right Button") (
    "Modem"
);
resource 'STR' (-4093, "Left Button") (
    "Printer"
);
resource 'STR' (-4091, "List label") (
    "Select Speed and Click on a Port."
);
resource 'STR' (-8191, "Spool File Name") (
    "Print File"
);
resource 'DITL' (-8191, "Job dialog template") (
    (
        (8, 321, 28, 381),
        Button (
            enabled,
            "OK"
        ),
        (9, 395, 29, 455),
        Button (
            enabled,
            "Cancel"
        ),
        (6, 6, 23, 127),
        StaticText (
            disabled,
            Version
        ),
        (30, 5, 46, 89),
        StaticText (
            disabled,
            "Page Range:"
        ),
        (30, 93, 45, 138),
        RadioButton (
            enabled,
            "All"
        ),
        (30, 140, 46, 200),
        RadioButton (
            enabled,
```

```

        "From:"
    ),
    {30, 205, 46, 237},
    EditText (
        disabled,
        ""
    ),
    {30, 242, 46, 270},
    StaticText (
        disabled,
        "To:"
    ),
    {30, 272, 46, 304},
    EditText (
        disabled,
        ""
    ),
    {55, 5, 71, 89},
    StaticText (
        disabled,
        "Copies:"
    ),
    /* [11] */
    {55, 95, 71, 127},
    EditText (
        disabled,
        "1"
    ),
    {55, 145, 71, 200},
    StaticText (
        disabled,
        "Feed:"
    ),
    {55, 205, 70, 303},
    RadioButton (
        enabled,
        "Continuous"
    ),
    {55, 305, 71, 425},
    RadioButton (
        enabled,
        "Sheet Feed"
    )
}
);
resource 'DITL' (-8192, "Style Dialog Template") (
    {
        {44, 390, 64, 450},
        Button (
            enabled,
            "OK"
        ),
        {10, 390, 30, 450},
        Button (
            enabled,
            "Cancel"
        ),
        {10, 4, 30, 124},
        StaticText (
            disabled,
            "DMP-110 v1.0"
        ),
        {37, 66, 53, 120},
        StaticText (
            disabled,
            "Style:"
        ),
        {0, 0, 0, 0},
        UserItem (
            enabled
        )
    }
);
resource 'DITL' (-8190, "Next Page Template") (

```

```

    {
        {30, 60, 46, 140},
        Button (
            enabled,
            "Done"
        ),
        {30, 160, 46, 240},
        Button (
            enabled,
            "Stop"
        ),
        {8, 8, 24, 120},
        StaticText (
            enabled,
            Version
        ),
        {8, 120, 24, 265},
        StaticText (
            enabled,
            "Insert Next Sheet"
        )
    }
);
resource 'DLOG' (-8191, "Job Dialog") (
    {52, 16, 136, 492},
    dBoxProc,
    invisible,
    noGoAway,
    0x1,
    -8191,
    "Job"
);
resource 'ICON' (-4080, "DMP-110") (
    "$*0000 0000 0000 0000 0000 0000 0000 0000"
    "$*0000 0000 0000 0000 0000 0000 007F 8000"
    "$*0040 C000 0040 A000 0040 9000 0040 F800"
    "$*0040 0800 0040 0800 0040 0800 0040 0800"
    "$*0040 0800 0040 0800 03FF FFC0 04AA AC20"
    "$*04D5 55A0 04AA ADA0 027F F840 0200 0040"
    "$*01FF FF80 0008 0400 0008 0400 0008 0400"
    "$*1E08 0E00 1FF0 1F00 1E00 1F00 0000 0A" );
resource 'DLOG' (-8192, "Style Dialog") (
    {30, 20, 113, 488},
    dBoxProc,
    invisible,
    noGoAway,
    0x1,
    -8192,
    "Stl" );
resource 'DLOG' (-8190, "Next Page Box") (
    {48, 51, 100, 330},
    dBoxProc,
    -1,
    noGoAway,
    0x0,
    -8190,
    "Next Page" );
/* Print Record provides 60 dpi in the printing portRect as
 * seen by application, and scales the image up to 120 dpi.
 * It is useful scaling up BitMaps to double image size, or
 * word processing applications where text justification is
 * needed. */
resource 'PREC' (0, "Default Scale") (
    3,
    0,
    0,
    60,
    60,
    {0, 0, 630, 480},
    {-15, -15, 645, 495},
    512,
    1320,
    1020,
    0,

```



```

1, /* bU10ffset */
1, /* bU1Shadow */
0, /* scanTB */
0, /* unused */
1,
9999,
1,
0,
1,
0,0,
0,0,
0,
(
3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
);
/* Print Record gives a printing port resolution of 80 dpi,
* which is close to ImageWriter res & therefore close
* to what most Macintosh applications expect to see. The
* problem with this 'PREC' is that final image is scaled
* by a non-integral factor, & so text justification in word
* processing programs fails by a few bits width.*/
resource 'PREC' (4,"Compatible") (
3,
0,
0,
80,
80,
(0, 0, 840, 640),
(-20, -20, 860, 660),
512,
1320,
1020,
0,
1,
0,
0,
120,
120,
(0, 0, 1260, 960),
/* TPrXInfo */
120, /* rowBytes */
64, /* vertical dots */
959, /* horizontal dots */
7680, /* size of bit image */
20, /* bands per page */
0, /* bPatScale */
1, /* bU1Thick */
1, /* bU10ffset */
1, /* bU1Shadow */
0, /* scanTB */
0, /* unused */
1,
9999,
1,
0,
1,
0,0,
0,0,
0,
(
4,-1,-1,0,0,0,0,0,0,0,0,0,0,0,0,0,0
);
resource 'STR#' (-4080, "Baud Rates") (
"300",
"600",
"1200",
"1800",
"2400",
"3600",
"4800",
"7200",

```

```

"9600",
"19200",
"57600"
);
resource 'DITL' (-4078, "ATalk Items") (
(
{139, 121, 159, 181},
Button (
enabled,
"OK"
),
(54, 11, 125, 294),
StaticText (
enabled,
"This printer cannot use the printer port now "
"because AppleTalk is active. Please either "
"turn off AppleTalk, use the modem port "
"instead, or choose another printer."
),
(9, 235, 41, 267),
Icon (
enabled,
-4080
)
)
);
resource 'DITL' (-4079, "Sys41 Items") (
(
{90, 137, 110, 197},
Button (
enabled,
"OK"
),
(56, 21, 76, 308),
StaticText (
enabled,
"This Printer requires System 4.1 or newer!"
),
(11, 274, 43, 306),
Icon (
enabled,
-4080
)
)
);
resource 'ALRT' (-4079, "Sys41Alert") (
{40, 36, 164, 370},
-4079,
(
OK, visible, sound1,
OK, visible, sound1,
OK, visible, sound1,
OK, visible, sound1
)
);
resource 'ALRT' (-4078, "ATalk is on!") (
{48, 38, 216, 342},
-4078,
(
OK, visible, sound1,
OK, visible, sound1,
OK, visible, sound1,
OK, visible, sound1
)
);

```

Listing: NPWFinal.r

```

/* This is the "Stick-it-all-together" Rez file. */
/* file is part of the DMP-110 printer driver for the Mac
* series of computers. */
INCLUDE "PACK";
INCLUDE "PDEF0";
INCLUDE "PDEF4";
INCLUDE "PDriver";
INCLUDE "dmp-110.rsrc";

```



C Workshop

Writing INITs in THINK C



*J. Peter Hoddie
Palo Alto, CA
Volume 5, Number 10*

Writing INITs Using THINK C Introduction

THINK C provides a convenient and powerful environment for creating INITs. INITs are code resources, stored in the System Folder, that are automatically run at start up. THINK C allows the programmer to easily access global variables from an INIT, as well as providing an inline assembler to deal with situations when C code alone may not be sufficient. This article will present a simple, but complete, INIT created with THINK C. In addition, the details of how THINK C handles code resources will be presented along with some general rules for writing INITs. While the examples presented here are written in THINK C, many of the techniques, tips, and discussions apply to writing INITs in any language.

Building an INIT in THINK C is the same as building any other type of code resource such as CDEF, LDEF, WDEF, or FKEY. You use "Set Project Type..." to indicate that the project is a Code Resource, and you set the "File Type" field to INIT. To be safe, you should use the Attributes field to set the lock bit and reset the purge bit so that your INIT code won't get moved or purged unexpectedly. Set the System Heap bit to ensure that the INIT is loaded into the System Heap, not the Application Heap. If you do not select the "Custom Headers" option THINK C installs some header code that loads A0 with the address of your code resource and then branches to your "main" routine. For the purposes of this article, the default THINK C header will suffice.

THINK C comes with a set of macros contained in the file "SetUpA4.h" which take care of the necessary details so that you can access global variables from within your INIT. THINK C maintains your global variables as part of the code resource that makes up the INIT, placing the global variables at the end of the code resource. Since THINK C generates code that accesses these variables with a word as an offset, the maximum combined length of the code and all global variables is 32K. The "SetUpA4.h" file generates a small portion of code, so it should only be included once.

INIT Installation

Most INITs consist of two main parts. The first is the installation code which is executed at start up time. This code usually patches a few traps and returns control to the system. The second portion of the INIT is the code that will be executed when the patched trap is called. Some INITs do not patch any traps but may instead install VBL tasks or load various drivers.

The Main procedure in your THINK C project is the installation code. It is called immediately after your INIT is loaded. Main must perform several tasks so that the trapped patches will function properly. The first task is to remember the address that

the INIT resource was loaded at so that global variables may be accessed. The following code fragment illustrates the essential elements.

```
*include <SetUpA4.h>
Handle myINITHandle
main()
{
    Ptr myINITPtr;

    asm {
        move.L A0,myINITPtr;
    }
    RememberA0();
    SetUpA4();
    myINITHandle = RecoverHandle( myINITPtr );
    DetachResource( myINITHandle);
    ...
    RestoreA4();
}
```

The very first line of code in Main stores the contents of register A0 in the local variable myINITPtr. This value must be stored in a local variable, as global variables are not yet available at this point in the code. Once this is done, the macros RememberA0 and SetUpA4 from the "SetUpA4.h" file are called. RememberA0 causes the current value of A0 to be stored away (in a memory location reserved in the code generated "SetUpA4.h"), and SetUpA4 causes A4 to be loaded with the address of our INIT code resource. THINK C generates global variable references relative to A4 for code resources, so global variables may be referenced after the call to SetUpA4. The next step is to use the address of the INIT code resource that we saved (which is a separate copy from the one saved by the call to RememberA0) to recover the handle to our code resource and save that away in the global variable myINITHandle.

INITs are discussed in Inside Macintosh IV-256 which states that on entry to your INIT code the operating system "saves all registers and places the handle to your 'INIT' resource in register A0." This can lead to some confusion because the default THINK C header for a code resource places a pointer to your code resource in register A0. This destroys the handle to your INIT resource that the operating system put into A0, thus the call to RecoverHandle is required to get the handle of the INIT code resource.

The next line performs a DetachResource on the INIT code. This is necessary so that your INIT will survive beyond system start up. When you return from your INIT to the operating system, the resource file of your INIT is closed causing the INIT code resource to be purged from memory. Calling DetachResource forces the Resource Manager to forget that it ever knew about the INIT resource, so it is not purged.

Following the DetachResource call is any installation code for the INIT. This is where trap patches (discussed below) are placed. Ending the installation procedure is a call to RestoreA4 which simply restores the value to A4 that was present when SetUpA4 was called. RememberA0 should only be called once in your entire INIT, at the start of the installation procedure. SetUpA4 and RestoreA4 should always be called in pairs, in the same procedure. SetUpA4 stores the old value of A4 on the stack so that if you call RestoreA4 in a function other than the one that contained the SetUpA4 you will likely die a quick and violent death.

If you did not set the Lock bit in the "Set Project Type..." dialog, you should lock the INIT with a call such as

```
HLock(myINITHandle);
```

after the RecoverHandle call in your Main procedure. If you don't make sure that your INIT is locked and unpurgeable, it could very well be unexpectedly moved or removed by the Memory Manager.

Accessing the INIT File After Start Up

As mentioned above, at start up time, after your INIT returns from its Main procedure back to the operating system, the INIT's resource files are closed. Unfortunately, often times your INIT may need to access its resource or data fork at some later time. As good Macintosh developers, we all want our users to be able to rename their INITs. So it now becomes necessary to hunt down the name of our INIT file during start up and store it away to later to access the file. This can be accomplished with the following function which stores the name of the INIT file in the variable "name" passed to it. The passed variable should probably be a global variable.

```
findMyName(name)
Str255 name;
{
    FCBPBR p;

    p.ioCompletion = 0;
    p.ioRefNum = CurResFile();
    p.ioVRefNum = 0;
    /* next line is required, but why? */
    p.ioNamePtr = (StringPtr)name;
    PBGetFCBInfo(&p, false);

    BlockMove(p.ioNamePtr, &name,
        1 + *(char *) (p.ioNamePtr) );
}
```

The above function should be called very early in your installation code as it relies on the fact the the current resource files is the INIT. This will not be the case if you have opened any other resource files. If you are going to the trouble of saving the name of your INIT file you might also consider saving the current volume reference number. Through System 6, all INITs are stored in the System folder. The volume reference number of the System folder is easily found using the SysEnvirons call described in IM V-5. It is possible in the future that Apple will create an "INIT Folder" or that a developer will release a product that allows users to keep INITs in a separate folder. Thus in self defense, it is probably a good idea to store the current volume

reference number at the same time you save the file name of your INIT if you plan to access your INIT file again. The current volume reference number can be found using the call GetVol as described in IM II-89.

A secondary method of accessing resources in your INIT file is to load them into memory at start up time, and perform a DetachResource on each so they will not be purged. This solution is fine for a few small resources. However, memory will be unnecessarily tied up if you load many resources. Furthermore, you have no way of permanently modifying resources if they are all loaded into memory. A reasonable approach would be to keep any small resources that you access regularly in memory all the time, and only access the INIT resource file for rarely used or very large resources.

If you are loading resources at start up time that you intend to use at a later time, make sure that you set the System Heap bit on each one so that they are not loaded into the application heap. Furthermore, remember to call DetachResource on each resource, or they will be purged when your INIT's resource file is closed by the operating system on return from installation.

Patching Traps - Introduction

Patching traps is a very powerful means of altering system behavior. While it is very powerful, it is also rather easy to make mistakes. A trap patch is installed by means of a call to NSetTrapAddress, usually after saving the old address of that trap by calling NGetTrapAddress. Traps are not usually completely replaced by a patch. Instead, the data passed to the trap is intercepted before the actual trap gets to it, or by modifying the data after the trap is finished, but before it returns to the calling program. In effect, installing a patch trap puts another level of code between the calling program and the various operating system managers. Because trap patches may get called very often (for example a patch on GetNextEvent, common in most screen saver and macro packages) they should execute quickly, so as not to slow the system down.

There is no simple formula for writing a function to behave as a trap patch. You must read the Inside Macintosh description of the trap you wish to patch with great care. There are two different kinds of traps, Operating System and Toolbox. Operating System calls pass their parameters in registers. Thus for Operating System traps, it is necessary to include some assembly language code. Toolbox traps use Pascal calling conventions, so they pass their parameters on the stack. THINK C can mimic Pascal calling conventions, so in most cases no assembly language is needed to patch Toolbox traps.

I know of no definitive way to tell a Toolbox trap from an Operating System trap except to look at its Inside Macintosh definition and see if it gives register usage. If there is no register description it is probably a Toolbox trap. Of course there are several parts Operating System traps that do not provide register information with the description. As an example, the new HFS calls were implemented through a single trap called HFSDispatch. A routine selector value is placed in register D0 to tell the operating system which routine is desired. The List Manager is a somewhat different example. All List Manager calls go through

one trap, "Pack0". A routine selector value is passed as a word on the stack to indicate which routine to actually call. However, the vast majority of calls are fairly straightforward to handle.

Patching a Toolbox Trap

As an example, we will now see how to patch BeginUpdate. This is trap A922, and is part of the Window Manager. Before going into the details of the trap patch code itself, it is necessary to actually install the patch. As mentioned above this is done with NGetTrapAddress and NSetTrapAddress. These are newer versions of SetTrapAddress and GetTrapAddress, and should always be used as they allow you to specify whether the trap being patched is an Operating System or a Toolbox trap. The following code fragment, combined with the outline of Main above, shows how to install patch on BeginUpdate.

```
#define BeginUpdateTrap 0xA922
long oldBeginUpdate;
main()
{
    /* start code */

    oldBeginUpdate = NGetTrapAddress
        (BeginUpdateTrap, ToolTrap);
    NSetTrapAddress (newBeginUpdate,
        BeginUpdateTrap, ToolTrap);

    /* end code */
}
```

The address of the original trap for BeginUpdate is stored in the global variable oldBeginUpdate, so that we can call the original routine from within the patch. Next, the address of the patch function, newBeginUpdate, is installed as the new address for BeginUpdate. The trap number of BeginUpdate is #defined to 0xA922 and used in the two calls. Some developers pass 0x0922, dropping the A. Inside Macintosh is not explicit about this matter, simply saying to pass the "trap number." However, in Apple's own INIT code, they usually pass the trap number with the preceding A.

The calling definition of BeginUpdate, as given in IM I-292, is:

```
BeginUpdate(wind: WindowPtr);
```

An equivalent function prototype for THINK C, to be included in the header of your program, would look like:

```
pascal void newBeginUpdate(WindowPtr w);
```

The function to patch BeginUpdate would have an outline as follows:

```
pascal void newBeginUpdate(w)
WindowPtr w;
{
    SetUpA4();
    /* pre-processing goes here */
    CallPascal(w, OldBeginUpdate);
    /* post-processing goes here */
    RestoreA4();
}
```

By using the prefix "pascal" we are telling THINK C to use Pascal calling conventions for this procedure. By giving the

function the exact same definition as the trap we are patching, we can easily access its parameters. In fact the variable w is the WindowPtr for window that is to be updated. Thus we can access the entire window structure and make any necessary modifications. The calls to SetUpA4 and RestoreA4 should begin and end the trap patch so that all global variables will be available throughout the patch.

THINK C provides the CallPascal function so that we can call Pascal style routines without reverting to assembly language. The CallPascal functions are described in detail in the THINK C User's Manual on pages 119-120. If the function that you are calling returns a value, other versions of the CallPascal function should be used, as described in the THINK C manual. In this use of CallPascal we are calling the original BeginUpdate routine that existed before we patched the BeginUpdate trap, so we simply pass the Window Pointer w and the address of the routine to CallPascal, and THINK C takes care of the details.

Patching An Operating System Trap

With a few exceptions, operating system traps do not store any parameters on the stack, rather they pass them in registers. Thus operating system trap patches must be handled differently than toolbox traps. Declare the function that will serve as the patch with no return value and no parameters in the header of the program, such as:

```
void osTrapPatch(void);
```

The beginning of the function then includes some assembly code to save the contents of the parameter registers. Calling the original trap must be done with assembly code as well, first restoring the appropriate parameters into the registers, and then calling the original trap. Furthermore, some operating system traps return values in a register (commonly D0). This value must be saved on return from calling the original trap, and placed in the appropriate register immediately before returning from the patch.

SetFileInfo (trap A00D, IM II-116) takes a parameter block pointer in register A0 and returns an error code as a word in register D0. The following example code assumes that the trap patch has been installed using the same procedure as shown above for BeginUpdate.

```
#define SetFileInfoTrap 0xA00D
void newSetFileInfo(void);
long oldSetFileInfo;

void newSetFileInfo()
{
    HFileParam *PBPtr;
    int saveD0;
    /* save original parameter block pointer */
    asm {
        move.L A0,PBPtr
    }
    /* make global variables available */
    SetUpA4();
    /* do any pre-processing */
    asm {
        /* set up the parameter block pointer */
        move.L PBPtr,A0
        /* get address of original trap */
        move.L oldSetFileInfo,A1
    }
}
```

```

/* call original trap */
jsr (A1)
/* save return value, error code */
move.W D0,saveD0
}
/* do any post-processing */
/* restore original value of A4 */
RestoreA4();
/* set-up return value */
asm {
    move.W saveD0,D0
}
}

```

Coping with the Trap Dispatcher

When any trap is called, the Trap Dispatcher is invoked first. The Trap Dispatcher performs some general house keeping before calling the actual routine. One task it performs that can cause problems, is the storing of the actual trap number that was called in register D1. Apple has never documented this fact, so it may very well change. In self defense you may want to use some assembly language to save the value of D1 on entry to your patch, and restore its value immediately before calling the original trap. In fact, if you want to be very safe, you may want to save and restore other registers as well. The outline of the code for saving and restoring D1 follows.

```

void patch()
{
    long saveD1;

    asm {
        move.L D1,saveD1
    }
    SetUpA4();
    /* any pre-processing */
    asm {
        move.L saveD1,D1
    }
    /* call the original trap */
    /* any post-processing */
    RestoreA4();
}

```

Note that saveD1 is a local variable because calling SetUpA4 could destroy the contents of D1. Before SetUpA4 is called only local variables are accessible. It is not necessary to restore the value of D1 before returning from the patch, although as a defensive measure you may want to.

Memory

When writing a trap patch you should check IM to see if the trap you are patching could move memory. If it can, then you can make all the Memory Manager calls you like. If the trap you are patching does not move memory according to IM then you better avoid calling the Memory Manager or any operating systems routines that could call the Memory Manager. If you move memory on an application when it isn't expecting it, you will have created a worthless INIT.

If you intend to allocate memory, it is important that you consider whether you need to allocate memory in the system heap or the application heap. Any memory that is allocated in an application's heap will be lost when that application quits,

whereas system memory will remain until the machine is reset. During start up, if you allocate memory it may end up going into a temporary application heap that is set up for INITs. This is fine for temporary work, but not if you need to allocate a block of memory for later use. There has always been a way to allocate memory in the system heap, but it requires the use of assembly language. With THINK C you can create a few simple procedures for allocating memory in the System heap. Apple describes these in Tech Note #219, "New Memory Manager Glue Routines," and their definitions are given below. Note that once the handle or pointer is allocated in the system heap, standard Memory Manager calls can be used to manipulate them. Further discussion of this can be found on page 151 of the THINK C User's Manual and IM II-32, 36.

```

Handle NewHandleSys(length)
long length;
{
    asm {
        move.L length,D0
        NewHandle SYS
        move.L A0,D0
    }
}

Ptr NewPtrSys(length)
long length;
{
    asm {
        move.L length,D0
        NewPtr SYS
        move.L A0,D0
    }
}

```

Calls such as NewHandleSys and NewPtrSys are used for grabbing a block of memory. However, they do not guarantee the memory will actually be available at start up time. In Inside Macintosh IV, a tricky method of expanding the System Heap is described. Fortunately, in Volume V a simpler solution is presented although its description is less than detailed. Simply include a "sysz" resource with ID=0 in your INIT file. This resource consists of a single long word that specifies the number of bytes of memory your INIT needs. The operating system will attempt to grow the system heap by that amount before loading your INIT. If your INIT does not contain a "sysz" resource, the system heap is expanded by 16K. When you return from your INIT's installation code, the system heap is compacted so that any memory that you didn't explicitly allocate is lost. All the "sysz" mechanism does is to make sure that a certain amount of memory is available for allocation. It does not actually allocate or reserve the memory for your INIT.

Warning

Tech Note #212, "The Joys of Being 32-Bit Clean," states "Make sure that any patch you do write is not a tail patch... You need to avoid tail patches because many of Apple's System Software patches check the return address on the stack to see who called them. If you write a tail patch, you defeat these checks and may cause things to break in strange and less than wonderful

ways.” Unfortunately, any trap patch written using the C function technique described here are effectively tail patches. However, I have yet to find a case where this actually causes a problem. Furthermore, in many cases it is impossible to obtain the desired result without writing a tail patch. It seems unlikely that Apple will declare war on tail patches in future Systems, as there are already tons of tail patches out there. However, it is possible that this could cause problems in some rare instances. Unfortunately, the alternative is to write lots more assembly code, and that isn’t a terribly appealing option.

Communicating with the User at Start Up

At installation time, many INITs may want to do something beyond simply installing a trap patch. In some cases, the INIT may need to interact with the user through a dialog. Apple is silent on this issue, although their own AppleShare INIT puts up an interactive dialog at start up. After some experimenting and disassembly I have come up with a way to handle dialogs at start up. The approach is very similar to what you do in a normal application. You must initialize various managers. The problem is that you should not initialize every manager, and there are some other matters to be taken care of. The sequence of calls below will set up the system so that the Dialog Manager may be used.

```
InitFonts();
InitWindows();
TEInit();
InitDialogs(0);
```

There are still a few of catches though. QuickDraw requires that A5 point to a set of its global variables. As an INIT, we don’t automatically get our own set of QuickDraw globals. Fortunately, the operating system provides us with a set we can use. All you have to do is load A5 from the global variable CurrentA5. This should be done before initializing any of the managers. The following line of assembly does the job.

```
asm (
    move.L CurrentA5,A5
)
```

There is no need to restore A5 to its original value as the operating system saves all registers before calling your INIT, and restores them on return.

One unfortunate side effect of calling InitWindows is that the entire screen is redrawn, so any icons drawn by INITs loaded before yours are lost. If you only want to draw an INIT at start up to notify the user that your INIT has been installed, get a copy of Paul Mercer’s ShowINIT code (which is available as a THINK C project) and use it. Almost everyone who puts up an icon at start up uses Mercer’s code or a variant of it. ShowINIT makes sure icons don’t overlap and so on. A really useful piece of code.

There are two low memory globals that have not been initialized at the time INITs are loaded which can cause problems. These are DeskHook and DragHook which are both set to -1 when INITs are loaded. If the operating system tries to branch through these vectors, an address error is generated and the system error handler steps in. Simply setting these two globals to zero before initializing any of the managers solves the problem.

```
DeskHook = 0;
DragHook = 0;
```

It is possible, although highly unlikely (since these globals are no longer used under MultiFinder), that a previous INIT or the operating system could have put a value into DeskHook and/or DragHook. For this reason, a more defensive method of handling these two globals is to test each to see if they contain an odd number. If so, set them to zero; otherwise assume that they are valid and leave them alone.

Uninstalling a Trap Patch

In most cases, once you have installed a trap patch you will not want to uninstall that trap patch. In some cases however, you may want to disable or remove the patch. Unfortunately, another INIT or application may have installed a patch on top of your trap patch. They have probably stored your entry address so that it can be called directly. Furthermore, some applications store the address of certain traps and call them directly to avoid the overhead of the Trap Dispatcher on each call. If you change the address of a trap after any other INIT or application has had the opportunity to execute, there is a chance that the system will be corrupted. In fact some INITs that users have found must load last make this mistake. The solution is not to change the trap address again, but to store a global state variable that indicates whether or not the INIT is active. The global variable can then be checked in each patch, and if the INIT is inactive, you can simply call the original trap without further intervention. There are more sophisticated ways to uninstall a patch which can even release most of the memory claimed by the INIT code, but they involve assembly language and/or self modifying code.

An Example

To illustrate some of the points made in this article, the following sample INIT has been provided. The INIT is a simple virus protection program. It intercepts two resource manager traps, ChangedResource and AddResource. Many viruses depend on these two calls to work. This INIT is not intended to be a full fledged virus protection program, but rather an example of how an INIT works. A list of resource types to watch is contained in the ‘ResT’ 256 resource. Initially this only contains CODE, INIT, and nVIR resources. This may be modified with ResEdit as desired. If the INIT detects a ChangedResource or AddResource call involving any resource type listed in the ‘ResT’ resource, it puts up a dialog warning the user about the pending action. The resource type, id number, name, and file name are displayed. The user may select OK to allow the operation to continue or Cancel to stop it. [This example is NOT meant for full virus protection; so don’t use it as such. This is an EXAMPLE on how to write an INIT; use it that was.-ed]

If the user holds down the mouse button when the INIT is loaded, it will not install itself. This feature can be useful when debugging the INIT and in cases where the user doesn’t want to load an INIT for a particular session. Many INITs currently use this approach. It would be easier for users if all INITs adopted such an approach.

The sample INIT illustrates the basic techniques described

here, but does not get into some of the more complex techniques in the interest of brevity and clarity. The INIT should be compiled with the "Precompiled Headers" option on.

Listing: example.c

```
#include <SetUpA4.h>

#define nil 0L

#define ChangedResourceTrap 0xA9AA
#define AddResourceTrap 0xA9AB

Handle queryDITL; /* DITL for unknown dialog */
Handle resourceTypes; /* handle of rsrc types to watch */
long oldChangedResource; /* addr of original ChangedResource */
long oldAddResource; /* address of original AddResource */
Str255 trash; /* needed due to Apple bug ? */

pascal void NewChangedResource(Handle h);
pascal void NewAddResource(Handle h, ResType rType, int id,
Str255 name);
Boolean userDialog();
Boolean inList(ResType type, Handle list);
Str255 *findFileName(int refNum);
void main(void);

pascal void NewChangedResource(h)
Handle h;
{
    int id;
    ResType type;
    Str255 resName, num;
    Boolean ok;

    SetUpA4();

    ok = true;
    GetResInfo(h, &id, &type, &resName);

    if (inList(type, resourceTypes) )
        ok = userDialog("\pAttempt to Change Resource",
            type, id, &resName, HomeResFile(h) );

    if (ok)
        CallPascal(h, oldChangedResource);
    else
        ResErr = resAttrErr;

    RestoreA4();
}

pascal void NewAddResource(h, rType, id, name)
Handle h;
ResType rType;
int id;
Str255 name;
{
    Boolean ok;

    SetUpA4();

    ok = true;

    if (inList(rType, resourceTypes) )
        ok = userDialog("\pAttempt to Add Resource",
            rType, id, name, CurResFile(h) );

    if (ok)
        CallPascal(h, rType, id, name, oldAddResource);
    else
        ResErr = addResFailed;
}
```

```
RestoreA4();
}

Boolean userDialog(message, type, id, resName, file)
Str255 *message;
ResType type;
int id;
Str255 *resName;
int file;
{
    GrafPtr oldPort;
    Handle tempH;
    DialogPtr d;
    int i;
    Rect r;
    Str255 num;

    GetPort(&oldPort);

    tempH = queryDITL;
    HandToHand(&tempH);
    SetRect(&r, 90, 68, 444, 226);
    d = NewDialog(nil, &r, nil, true, 1, -1, false, nil, tempH);

    SetPort(d);
    MoveTo(20,20);
    DrawString(message);
    MoveTo(20,40);
    DrawString("\pResource Type: ");
    DrawText(&type, 0, 4);
    MoveTo(20,60);
    DrawString("\pResource ID: ");
    NumToString((long)id, num);
    DrawString(num);
    MoveTo(20,80);
    DrawString("\pResource Name: ");
    DrawString(resName);
    MoveTo(20,100);
    DrawString("\pFile Name: ");
    DrawString( findFileName( file ) );

    do {
        ModalDialog(nil, &i);
    } while ( i != 1 && i != 2 );

    DisposDialog(d);

    SetPort(oldPort);

    return( i == 1 );
}

Boolean inList(type, list)
ResType type;
Handle list;
{
    int len;
    ResType *resPtr;

    len = GetHandleSize(list) >> 2;
    resPtr = (ResType *)list;
    while ( len-- )
        if (*resPtr++ == type)
            return(true);
    return(false);
}

Str255 *findFileName(refNum)
int refNum;
{
    FCBPBRec p;

    p.ioCompletion = 0;
    p.ioRefNum = refNum;
```



```

p.ioFCBIndx = 0;
p.ioVRefNum = 0;
p.ioNamePtr = (StringPtr)trash;
PBGetFCBInfo(&p, false);

return((Str255 *)p.ioNamePtr);
}

/*This block is called once. It saves the pointer
to this code resource, and installs the patch.*/
void main()
{
    Handle myHandle;
    Ptr myPtr;
    SysEnvRec world;
    Str255 *namePtr;

    asm {
        move.l A0, myPtr
    }
    RememberA0();
    SetUpA4();
    if(!Button()) {
        myHandle = RecoverHandle(myPtr);
        DetachResource(myHandle);

        resourceTypes = GetResource('ResT', 256);
        DetachResource(resourceTypes);

        queryDITL = GetResource('DITL', 256);
        DetachResource(queryDITL);

        oldChangedResource =
NGetTrapAddress(ChangedResourceTrap, ToolTrap);
NSetTrapAddress(NewChangedResource, ChangedResourceTrap, ToolTrap);

        oldAddResource =
NGetTrapAddress(AddResourceTrap, ToolTrap);

```

```

NSetTrapAddress(NewAddResource, AddResourceTrap, ToolTrap);
    }
    RestoreA4();
}

```

Listing: Example.r

```

resource 'DITL' (256, sysheap) (
    /* [1] */
    (129, 270, 149, 330),
    Button (
        enabled,
        "OK"
    ),
    /* [2] */
    (129, 189, 149, 249),
    Button (
        enabled,
        "Cancel"
    )
);

resource 'sysz' (0) (
    0x0800
);

resource 'ResT' (256, sysheap) (
    /* [1] */
    'nVIR',
    /* [2] */
    'INIT',
    /* [3] */
    'CODE'
)
);

```



MIDI Mac

Apple's MIDI Manager



Don Veca
San Jose, CA
Volume 5, Number 11

Introduction

The MIDI Manager is Apple's new standard MIDI tool set. It enables developers to cleanly transfer MIDI data to and from synthesizers and other concurrently running applications. It supplies a complete MIDI tool set which is as powerful as any existing implementation, yet strictly adheres to *system* guidelines. Included with the MIDI Manager are the *Apple MIDI Driver* and *PatchBay*.

The Apple MIDI Driver interprets all incoming and outgoing MIDI messages and handles all serial port I/O. This includes various modes of time code generation and concurrent asynchronous serial port communication. PatchBay is Apple's graphical user interface for the MIDI Manager. Supplied as a utility application and a Desk Accessory, PatchBay enables users to intuitively route MIDI data and time code to and from the Apple MIDI Driver as well as throughout multiple applications running under Multifinder. The MIDI Manager proper, however, supplies all internal communications and supplies a thorough and very powerful MIDI tool set.

Main Concepts

From the music software developer's point of view, the main components of the MIDI Manager are **clients** and **ports**. A client is any program that uses the MIDI Manager's facilities. Although an application usually needs only one client, it can actually have several. Each client can, and usually does, create multiple ports which are basically unidirectional streams of MIDI information. Each port can be either a *time port*, which provides the MIDI Manager's timing facilities, or a *data port*, which is used to read or write MIDI *packets*. The data ports are further divided into *input* and *output* ports. A client's ports can be connected to one of its own ports or to any other client's ports; however, input ports can only be connected to output ports (and vice versa). Time ports enable port synchronization: when a time port's clock ticks, all clocks of all ports connected to it tick synchronously. Data ports send and receive MIDI data such as note-on/note-off or system exclusive messages. Input and output ports respectively receive and send MIDI data to the ports of MIDI drivers or other MIDI Manager compatible applications running concurrently under Multifinder. We will now illustrate the proper use of the MIDI Manager through the explanation of a sample application called MIDIArp.

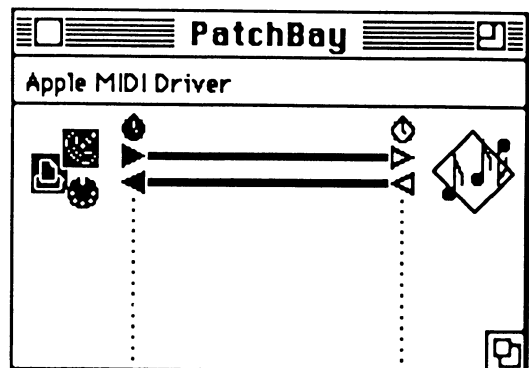
The MIDIArp Demo Application

MIDIArp is a simple arpeggiator program [tone generator for those who are not blessed with a huge vocabulary like Don's. -ed] that illustrates the use of the MIDI Manager's data and timing facilities. MIDIArp simply reads in *note-on* data and

arpeggiates it until corresponding *note-off* data is received. The basic flow of MIDIArp is quite simple: after MIDIArp initializes the various Macintosh Managers, it signs into the MIDI Manager, calls a routine to set up its time, input, and output ports, pops up its main dialog box, and cycles through its main event loop. The main event loop simply checks for user (console) input, and adjusts its arpeggio direction and speed parameters accordingly. Once the *quit* button is selected, MIDIArp signs out from the MIDI Manager and terminates. During the main event loop, however, MIDIArp continually arpeggiates its MIDI input at interrupt level via its **readHook** and **timeProc** routines.

MIDIArp.h

MIDIArp includes a header file, *MIDIArp.h*, which contains its main data structures and symbolic constants. In addition to the standard user interface constants for menu and dialog resource ID's, constants are defined to identify our own resource of type 'port'. We define three such 'port' resources: one for each MIDI Manager port we intend to use. This is necessary in order to save the state of our *patch* between each launch of the application. For readability, several MIDIArp constants are defined followed by several constants local to the MIDI Manager itself. Specifically, we define our **client** and 'ICN#' resource ID, both of which are used to *sign in* to the MIDI Manager. The **client** ID is a four-byte OSType (and by convention, although not by necessity, our **client** ID is used as our application **signature**). We then define our port ID's which are also of type OSType. The actual ports will be displayed by PatchBay from top to bottom in ascending alphabetical order. (It is recommended that time ports be displayed above the data ports that are synchronized to them, and that an application's input and output ports be displayed in the reverse order of that used by the Apple MIDI Driver.)



After several MIDI Manager parameters are symbolically defined, we define our main data structures. The first data

structure that we will need is a *NoteInfo* record which contains fields to store the MIDI channel, key number, and key velocity of each incoming *note-on* message. The *NoteInfo* record is itself an individual field of the main MIDIArp data structure *ArpParams*. The *ArpParams* structure contains various fields of information concerning the current state of the MIDIArp client. In particular, it contains a field called *Locked*, which is used to prevent the structure from being modified while it is in use. Incoming MIDI data (notes that MIDIArp is currently arpeggiating) are stored in an array of *NoteInfo* records called *NoteTbl*. Other fields hold information about things such as tempo and current arpeggiation pattern (e.g., whether we are currently going up or down, etc.). *NextNoteOn* is a field used to keep track of the exact time the next note is to be played. Finally, the *ArpParams* structure allows the storage of each port's reference number.

MIDIArp.c

The main source file for MIDIArp is *MIDIArp.c*. Here we first define several global variables, including an *ArpParams* record called *ArpGlobals*, a variable to hold our current arpeggiation speed ID, and a flag (*GManualPatch*) to indicate whether the current port configuration (patch) has been set up by a (previously saved) *PatchBay* patch or needs to be reconfigured by MIDIArp itself (based on its last configuration). The *main()* routine of MIDIArp calls *InitThings()* to initialize the Macintosh Managers, *ArpInit()* which signs-in to the MIDI Manager and sets up our ports, *StartDialog()* to bring up the main dialog box with default settings, and then *RunDialog()* to handle events. When the *Quit* button is finally hit, *ArpClose()* is called to sign-out from the MIDI Manager, and the main dialog box is shut down through a call to *StopDialog()*. *InitThings()*, in addition to initializing the various managers of the Macintosh, sets up a standard menu bar and seeds the random number generator for random arpeggiation. *ArpInit()*, on the other hand, completely sets up MIDIArp's MIDI Manager environment. In order to use the MIDI Manager we must first make sure that it is currently installed. This is achieved through a call to *SndDispVersion()* (sound dispatch version). Given the constant *midIToolNum*, *SndDispVersion()* returns the version of the currently installed MIDI Manager or zero if the MIDI Manager is not installed. Once we have concluded that the MIDI Manager is installed, we must sign in to the MIDI Manager (before we make any other calls) by calling *MIDISignIn()*. We pass as arguments to *MIDISignIn()* our client ID, our client reference constant, a handle to our 'ICN#' resource, and our client name string. The client ID is used for future MIDI Manager calls and allows other clients, such as *patchers*, to get information about us. The client reference constant, or *refCon*, is a general purpose parameter that is only really needed by certain types of applications (such as *device drivers*). The handle to the 'ICN#' resource and the client name string are passed to allow *PatchBay* (or any other clients) to display them via *MIDIGetClientIcon()* and *MIDIGetClientName()*, respectively.

The next thing we do in *ArpInit()* is add our time, input, and output ports via *MIDIAddPort()* and connect them accordingly. The tricky part is determining whether the ports will automati-

cally be connected via a saved *PatchBay* patch, or whether we must connect them ourselves. Therefore, we set the global flag *GManualPatch* to true before we add any ports allowing us to determine by the return value of *MIDIAddPort()* whether or not we need to connect the ports ourselves. If *MIDIAddPort()* returns *midIVConnectMade* (virtual connection resolved), then we know that the ports were *virtually* connected by someone else via *MIDIConnectTime()* or *MIDIConnectData()*. In this case we simply set *GManualPatch* to false indicating that we do *not* have to *manually* patch together the ports. If *GManualPatch* is still true after all ports have been added, then we will call *PatchPorts()* to do our own patching.

The first port we add in *ArpInit()* is our time port or *time base*. We do this by calling *MIDIAddPort()*. This call will create a new port with attributes as described in an *InitParams* data structure; therefore, we must first set one up. The *init* record contains various information including items such as the port ID, the port type (*time*, *input*, *output*, *invisible time*) the port's *readHook*, the time format, port name, etc., and a port reference constant (*refCon*). If the port's *readHook* routine is to be called at interrupt level, then the *refCon* can be used to store the contents of the application's A5 register for global variable access. The same basic strategy is used for creating the input and output ports; however, MIDIArp by default wants its data ports to be *sync'd* to its time port. So before we actually add the port, we set the *timeBase* field of the *InitParams* record to our time port's reference number (previously obtained when we added our time port).

Now that all the ports have been added, we check *GManualPatch*, and, if it's still true, we call *PatchPorts()* to patch the ports as they were when we last quit. To reconfigure our port connections, we must read in the 'port' resource of each port. The port resource is nothing more than the saved result of *MIDIGetPortInfo()* from our last session. *MIDIGetPortInfo()* returns a handle to a record containing the port type, time base of the port, and a list of all its connections.

To reconfigure our time port, we first check its port *info* record (of its 'port' resource) to see if we should be *sync'd* to another client's time base. If so, we call *MIDIConnectTime()*, connecting the external time port to our time port (slaving us to it). If the result of *MIDIConnectTime()* is *midIVConnectErr*, then the external port's owner is not currently signed in; otherwise, we must set our sync mode to *externalSync*. Next, we check to see if we are supposed to be the time base of one or more external ports. If so, we connect our time port to each.

It's a little less complicated to reconfigure our input and output ports. All we have to do is connect our input and output port to each of the ports listed in the *PortInfo* record contained in the corresponding 'port' resource. The last thing we need to do in *ArpInit()* is start our time port's clock by calling *MIDIStartTime()*. Now that everything is set up, we simply cycle through our main event loop, waiting for the MIDI Manager to call our *readHook* routine with incoming MIDI data.

In addition to handling user events, the main event loop periodically checks to see if an external time base has suddenly been connected to us. This is achieved by first detecting that

“something in MIDI Manager world has changed,” and then by checking our time port’s info record to see if the MIDI Manager reports that we currently have an external time base. This situation arises if another client attempts to connect themselves to us or if a user of a *patcher* program (such as Patchbay) manually connects an external time port to ours. We check for either event in our main event loop by calling **MIDIWorldChanged()**. If **MIDIWorldChanged()** returns **true**, then we know that something in the current world has changed; however, this could be caused by a client signing in or out, a port being added or removed, or a connection being made or removed. Therefore, we must call **MIDIGetPortInfo()** on our time port and check whether the returned port information record reports that we currently have an external time base. If so, we then set our time port to external sync via **MIDISetSync()**; otherwise, we set it back to **midiInternalSync** (in the case that we were currently in external sync mode). The only thing left now is to get and process incoming MIDI data; this operation is handled at interrupt level through use of a **readHook** and a **timeProc** routine.

Before jumping into the details, a general explanation of the interrupt level control flow of MIDIArp’s **readHook** and **timeProc** is needed. The **readHook** is called by the MIDI Manager whenever an incoming message becomes “current.” When the first note of an input sequence is sent to MIDIArp’s input port, the MIDI Manager calls MIDIArp’s **readHook** which then copies it into the application’s note table buffer and calls the **timeProc** to take care of the output. When called, the **timeProc** determines the next note in the note table to be played, writes out the selected note, and then tells the MIDI Manager exactly when to call the **timeProc** for the next output. In other words, the only time the MIDIArp application calls its **timeProc** is from within its **readHook**, and this is done only when it receives the first note of an input sequence. After that, the **timeProc** itself is responsible for setting up its next wake-up.

As summarized above, MIDIArp’s **readHook ArpReader()** reads all incoming data, and starts a series of **timeProc wake-ups** which create the arpeggiation effect. As you may recall, the **refCon** field of the initialization record of each port was set to point to our application’s global variables and passed as an argument to **MIDIAddPort()** when the port was originally created; additionally, the **readHook** field of the input port’s initialization record was set to the address of **ArpReader()**. When the MIDI Manager calls a port’s **readHook**, it passes two parameters: the next “current” **MIDIPacket**, and the port’s **refCon** value. Because the MIDI Manager calls **ArpReader()** at interrupt level, the **readHook** first sets up our *A5 world*. This is achieved by calling the System routine **SetA5()** with the **refCon** parameter. Since we will be modifying the note table in the **ArpGlobals** record, we must check whether any other routine is currently reading it by checking the **Locked** field. If it is locked, we simply return **midiKeepPacket**, which tells the MIDI Manager to save the new packet — we’ll get it later. The packet is of interest to MIDIArp only if it is does not contain a MIDI Manager system specific message (indicated in the *message type* field of the **flags** byte), and does contain a note-on message. If these

conditions are satisfied, then we set the return value to **midiMorePacket**, telling the MIDI Manager that we now have this packet and that we want the next one.

Before we actually return, however, we process the current packet. If the status byte in the packet indicates a note-on message, then we copy it into our note table and increment our current note count. If our note count is now equal to 1 (i.e., this is the first note of an arpeggio), then we call **ArpTimeProc()** to initiate an arpeggiation. (**ArpTimeProc()** is the routine that actually writes out the MIDI data.) If the status byte in the packet is a note-off message, however, we locate the matching note in the note table; and, if it’s still a valid note (it didn’t get “stolen”), then we simply delete it from the note table and decrement the table’s current note count **NumNotes**. (A note can get “stolen” if there are 32 notes in our note table and a new note that is lower than the highest note is inserted.) Finally, we restore the system’s *A5 world* and return.

As mentioned above, we call our **timeProc ArpTimeProc()** from our **readHook ArpReader()** when we get the first note of an arpeggiation series. However, it is quite common for the MIDI Manager to call a **timeProc** at interrupt level (per an application scheduled event that will be illustrated below). This can be set up by calling the MIDI Manager routine **MIDIWakeUp()** with several arguments which include the reference number of a specific time port, a time, a period, and a pointer to the **timeProc**. In this case, the MIDI Manager calls the **timeProc** with the time port’s current time and **refCon**.

When our **readHook ArpReader()** calls our **timeProc ArpTimeProc()**, however, it passes it the time stamp of the current MIDI packet (which in this case is not used) and the **refCon** parameter that was passed to **ArpReader()**. (The **refCon** of each port points to MIDIArp’s global variables.) Once in control, the **timeProc** again sets up MIDIArp’s *A5 world* via its **refCon** parameter (because it may be called by the MIDI Manager as well as from within the **readHook**). We then lock the **ArpGlobals** structure so that **ArpReader()** won’t disturb it. If at this time there are no notes in our note table, then we simply cancel any pending **wake-ups** and return. Otherwise, we bump the note table index to the next note to be played. However, we must avoid the special case where the first note of an arpeggio may unintentionally be played twice in a row. (For example, the arpeggiation pattern is “Up,” two notes are struck “at same time,” and the first note received is higher than the second note. The first note received gets index 0, **noteTblIndex** is set to 0, and the note is played. When the next note is received, the previous note is moved to index 1, the new note is inserted into index 0, and the **noteTblIndex** is bumped to index 1, which is the note that gets played. This means that the first note will be perceived to be played twice!) To avoid this sequence of events, if the note that we played the last time the **ArpTimeProc()** was called (**LastNote**) is the same note as our new one, then we bump the note index once more. Now we can finally write out a note-on message by calling **MIDIWritePacket()**. And, because we know the duration of the note, we can write out its corresponding note-off by simply adjusting the time stamp and calling **MIDIWritePacket()** again with the same packet. The MIDI Manager will make sure that the

packet is actually written out at the specified time.

The last thing we do in *ArpTimeProc()* is schedule the time that the MIDI Manager should call *ArpTimeProc()* to write out the next note (this is what creates the arpeggio effect). We first set *NextNoteOnTime* to itself plus the value of *Tempo*; however, we want the *ArpTimeProc()* to be called (and write out the packet) soon enough before *NextNoteOnTime* such that the next call to *MIDIWritePacket()* will be before the packet's time stamp expires. Although we call *MIDIWritePacket()* with an accurately time-stamped packet, we call it early to avoid the chance of the note being received late due to processing time. The basic rule of thumb is to always write early, making sure that the time stamp (which is some time in the future) is accurate. Finally, we unlock the *ArpGlobals* structure, restore the system's *A5 world*, and return.

And that's it! The *readHook* continues reading incoming MIDI data, and the *timeProc* continues scheduling it to be written out. This continues until the user hits the *Quit* button. When this happens, we call *ArpClose()* to save our current *patch* configuration into the applications 'port' resource and *sign-out* from the MIDI Manager.

How to Put It All Together

Before running *MIDIArp*, however, the MIDI Manager must be installed. Installation consists of nothing more than moving the MIDI Manager and Apple MIDI Driver files into the *System* folder and restarting the Macintosh. Once the MIDI Manager is installed, *PatchBay* and any other MIDI Applications can be launched.

Under single *Finder*, use the *PatchBay DA* to connect a single application to and from the Apple MIDI Driver; under *MultiFinder* however, use the *PatchBay* application. Once *PatchBay* and one or more MIDI-Manager-compatible applications or drivers are up and running, the input, output, and time ports of the applications or drivers can be connected to themselves, other applications, and to the Apple MIDI Driver in any proper configuration.

When a MIDI instrument is connected to the Macintosh and *MIDIArp* is launched, the mouse can be used to connect *MIDIArp*'s output port to the Apple MIDI Driver's input port, and to connect the MIDI Driver's output port to *MIDIArp*'s input port. (The MIDI Driver's input port can be thought of as the Macintosh's output port and visa versa). In this configuration, *MIDIArp* will continue to arpeggiate (in various patterns and at various tempos) the note data output of one or more interfaced MIDI instruments or any other MIDI-Manager-compatible applications, desk accessories, or drivers.

Where to Get the MIDI Manager

The MIDI manager is currently available to developers through *APDA* and may possibly be shipped with future System Disks. The *MIDIArp* sample application (and several other samples) are included on the *APDA* disk as well. Although there are no MIDI-Manager-compatible third party applications available at the time of this writing, various Macintosh MIDI developers are currently working toward MIDI Manager compatibil-

ity, and it is likely that many third party application packages will become available in the near future.

[The full listing is not included due to licensing concerns; you may however get the MIDI manger and *PatchBay* on the source code disk for this issue, #50. -ed]

```
/*
   For commented, properly
   formatted, and complete
   source code, please refer
   to the MacTutor source
   listing disk for this issue.
*/
/** MIDIArp.h ***/
#define mainDialogID 2000
#define quitID 1
#define patternPromptID 2
#define patternUpID 3
#define patternDownID 4
#define patternUpDownID 5
#define patternDownUpID 6
#define patternRandomID 7
#define speedPromptID 8
#define speedVeryFastID 9
#define speedFastID 10
#define speedMediumID 11
#define speedSlowID 12
#define speedVerySlowID 13
#define arpAlertBoxID 12345
#define arpAboutAlertID 13554
#define portResType 'port'
#define timePortResInfoID 128
#define inputPortResInfoID 129
#define outputPortResInfoID 130
#define noteTblSize 32
#define goingUp 1
#define goingDown 0
#define speedVeryFast 50
#define speedFast 100
#define speedMedium 200
#define speedSlow 300
#define speedVerySlow 500
#define noteDuration (ArpGlobals.Tempo * 0.95)
#define arpClientID 'Marp'
#define arpIcon 128
#define timePortID 'Atim'
#define inputPortID 'Bin '
#define outputPortID 'Cout'
#define keyOnOffPacketSize9
#define stdPacketFlags 0
#define flagsTimeStampMask 0x7F
#define noTimeBaseRefNum 0
#define noClient ' '
#define noReadHook 0L
#define noTimeProc 0L
#define zeroTime 0L
#define zeroPeriod 0L
#define refCon0 0L
#define timePortBuffSize 0L
#define inputPortBuffSize 2048
#define outputPortBuffSize0L

typedef struct
( unsigned char Channel;
  unsigned char Note;
  unsigned char Velocity;
  unsigned char Dummy;
) NoteInfo;
typedef struct
( short Locked;
  long NoteIndex;
  LastNote;
  rection;
  NoteInfo NoteTbl[noteTblSize];
  short NumNotes;
  short ArpPattern;
  long Tempo;
  NoteInfo NoteInfo;
  short NextNoteOnTime;
  ArpDi-
```

```

short    InputRefNum; short    OutputRefNum;
short    TimeRefNum;
} ArpParams;

/*****
MIDIArp.c
*****/
// Include Standard Mac Headers
#include <MIDI.h>
#include "MIDIDefs.h"
#include "MIDIArp.h"

DialogPtr GMainDialog;
ArpParams ArpGlobals;
short    GCurSpeedID;
Boolean  GManualPatch;
Boolean  GDone = false;
char    GMIDIMgrVerStr[256];

main()
{
    InitThings();
    ArpInit();
    StartDialog();
    RunDialog();
    ArpClose();
    StopDialog();
}

void InitThings(void)
{
    FlushEvents(everyEvent, 0);    InitGraf(&qd.thePort);

    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs(NULL);
    InitCursor();

    {
        Handle MenuBar = GetNewMBar(menuBar);
        SetMenuBar(MenuBar);
        DisposHandle(MenuBar);
        AddResMenu(GetMHandle(AppleMenu), 'DRVr');
        DrawMenuBar();
    }

    Seed(); // Rand Num Generator
}

void ArpInit(void)
{
    MIDIPortParams Init;
    Handle    TheIconHndl;
    OSErr     TheErr;
    long      MIDIMgrVerNum;
    char      CStrBuf1[256];

    MIDIMgrVerNum = SndDispVersion(midiToolNum);
    if (MIDIMgrVerNum == 0)
    {
        ArpAlert("The MIDI Manager is not installed!
        Aborting...");
        ExitToShell();
    }
    else
    {
        StdMacVerNumToStr(MIDIMgrVerNum, GMIDIMgrVerStr);
        sprintf(CStrBuf1, "MIDI Manager Version %s",
            GMIDIMgrVerStr);
        ArpAlert(CStrBuf1);
    }
}

```

```

TheIconHndl = GetResource('ICN#', arpIcon);
TheErr = MIDISignIn(arpClientID, refCon0,
    TheIconHndl, "\pMIDIArp");
if (TheErr)
{
    ArpAlert("Trouble signing MIDIArp into MIDI
    Manager! Aborting...");
    ExitToShell();
}

GManualPatch = true;

Init.portID = timePortID;
Init.portType = midiPortTypeTime;
Init.timeBase = noTimeBaseRefNum;
Init.readHook = noReadHook;
Init.initClock.sync = midiInternalSync;
Init.initClock.curTime = zeroTime;
Init.initClock.format = midiFormatMSec;
Init.refCon = SetCurrentA5();
C2PStrCpy("TimeBase", Init.name);
TheErr = MIDIAddPort(arpClientID, timePortBuffSize,
    &(ArpGlobals.TimeRefNum), &Init);

if (TheErr == midiVConnectMade)
{
    GManualPatch = false;
}
else if (TheErr == memFullErr)
{
    ArpAlert("Not enough room in
    heap zone to add time
    port! Aborting...");
    MIDISignOut(arpClientID);
    ExitToShell();
}

Init.portID = inputPortID;
Init.portType = midiPortTypeInput;
Init.timeBase = ArpGlobals.TimeRefNum;
Init.offsetTime = midiGetCurrent;
Init.readHook = (Ptr) ArpReader;
Init.refCon = SetCurrentA5();
C2PStrCpy("InputPort", Init.name);
TheErr = MIDIAddPort(arpClientID, inputPortBuffSize,
    &(ArpGlobals.InputRefNum), &Init);

if (TheErr == midiVConnectMade)
{
    GManualPatch = false;
}
else if (TheErr == memFullErr)
{
    ArpAlert("Not enough room in heap zone to add input
    port! Aborting...");
    MIDISignOut(arpClientID);
    ExitToShell();
}

Init.portID = outputPortID;
Init.portType = midiPortTypeOutput;
Init.timeBase = ArpGlobals.TimeRefNum;
Init.offsetTime = midiGetCurrent;
Init.readHook = NULL;
Init.refCon = &ArpGlobals;
C2PStrCpy("OutputPort", Init.name);
TheErr = MIDIAddPort(arpClientID, outputPortBuffSize,
    &(ArpGlobals.OutputRefNum), &Init);

if (TheErr == midiVConnectMade){
    GManualPatch = false;
}
else if (TheErr == memFullErr)
{

```

```

    ArpAlert("Not enough room in heap zone to add output
port! Aborting...");
    MIDISignOut(arpClientID);
    ExitToShell();
}

if (GManualPatch)
{
    PatchPorts();
}

ArpGlobals.Locked = false;
ArpGlobals.NumNotes = 0;
ArpGlobals.ArPPattern = patternUpDownID;
ArpGlobals.Tempo = speedMedium;
GCurSpeedID = speedMediumID;

MIDIStartTime(ArpGlobals.TimeRefNum);
}

void StartDialog(void)
{
    GMainDialog = GetNewDialog(mainDialogID,
        NULL, (WindowPtr) -1);
    SetPort(GMainDialog);

    ChangeState(GMainDialog, ArpGlobals.ArPPattern ==
        patternUpID, patternUpID);
    ChangeState(GMainDialog, ArpGlobals.ArPPattern ==
        patternDownID, patternDownID);
    ChangeState(GMainDialog, ArpGlobals.ArPPattern ==
        patternUpDownID, patternUpDownID);
    ChangeState(GMainDialog, ArpGlobals.ArPPattern ==
        patternDownUpID, patternDownUpID);
    ChangeState(GMainDialog, ArpGlobals.ArPPattern ==
        patternRandomID, patternRandomID);

    ChangeState(GMainDialog, GCurSpeedID ==
        speedVeryFastID, speedVeryFastID);
    ChangeState(GMainDialog, GCurSpeedID == speedFastID,
        speedFastID);
    ChangeState(GMainDialog, GCurSpeedID ==
        speedMediumID, speedMediumID);
    ChangeState(GMainDialog, GCurSpeedID == speedSlowID,
        speedSlowID);
    ChangeState(GMainDialog, GCurSpeedID ==
        speedVerySlowID, speedVerySlowID);

    ShowWindow(GMainDialog);
    StdAdjustDLOGLocation(GMainDialog);
}

void RunDialog(void)
{
    OSErr TheErr = noErr;
    short ItemHit;
    EventRecordAnEvent;
    WindowPtr WhichWindow;
    Rect Boundry;
    MIDIPortInfoHdl PortInfoH;
    GrafPtr SavePort;

    while (!GDone) {
        if (MIDIWorldChanged(arpClientID))
        {
            PortInfoH = MIDIGetPortInfo(arpClientID, timePortID);
            if ((*PortInfoH).timeBase.clientID != noClient)
            {
                MIDISetSync(ArpGlobals.TimeRefNum, midiExternalSync);
            }
            else
            {
                MIDISetSync(ArpGlobals.TimeRefNum, midiInternalSync);
            }
        }
    }
}

```

```

    DisposHandle((Handle) PortInfoH);
}

if (WaitNextEvent(everyEvent, &AnEvent, updatePeriod, NULL))
{
    if ((AnEvent.what == keyDown) && (AnEvent.modifiers
        & cmdKey))
    {
        AdjustMenus();
        DoMenuCommand(MenuKey(AnEvent.message & charCodeMask)
            );
    }

    if (IsDialogEvent(&AnEvent))
    {
        if (AnEvent.what == updateEvt)
        {
            GetPort(&SavePort);
            SetPort((GrafPtr) AnEvent.message);
            StdHiliteButton(GMainDialog, quitID);
            SetPort(SavePort);
        }

        if (AnEvent.what == keyDown &&
            ((AnEvent.message & charCodeMask)
                == charEnterKey))
        {
            GDone = true;
        }
        else if (
            DialogSelect(&AnEvent, &GMainDialog, &ItemHit))
        {
            switch (ItemHit)
            {
                case quitID:
                    GDone = true;
                    break;

                case
                    patternUpID:
                case
                    patternDownID:
                case
                    patternUpDownID:
                case
                    patternDownUpID:
                case
                    patternRandomID:
                    SwitchRadio(GMainDialog, &(ArpGlobals.ArPPattern),
                        ItemHit);
                    break;

                case
                    speedVeryFastID:
                    ArpGlobals.Tempo = speedVeryFast;
                    SwitchRadio(GMainDialog, &GCurSpeedID, ItemHit);
                    break;

                case
                    speedFastID:
                    ArpGlobals.Tempo = speedFast;
                    SwitchRadio(GMainDialog, &GCurSpeedID, ItemHit);
                    break;

                case
                    speedMediumID:
                    ArpGlobals.Tempo = speedMedium;
                    SwitchRadio(GMainDialog, &GCurSpeedID, ItemHit);
                    break;

                case
                    speedSlowID:
                    ArpGlobals.Tempo = speedSlow;

```

```

SwitchRadio(GMainDialog, &GCurSpeedID, ItemHit);
break;

case
speedVerySlowID:
ArpGlobals.Tempo = speedVerySlow;
SwitchRadio(GMainDialog, &GCurSpeedID, ItemHit);
break;

default:
SysBeep(2);
break;
)
}
}
else {
switch (AnEvent.what)
{
case mouseDown:
switch (FindWindow(AnEvent.where,
&WhichWindow))
{
case
inMenuBar:
AdjustMenus();
DoMenuCommand(
MenuSelect(AnEvent.where));
break;

case
inSysWindow:
SystemClick(&AnEvent, WhichWindow);

break;
case
inContent:
if (WhichWindow != FrontWindow())
{
SelectWindow(WhichWindow);
AdjustMenus();
}
break;
case
inGoAway:
if (TrackGoAway(WhichWindow, AnEvent.where))
{
GDone = true;
}
break;
case
inDrag:
SetRect(&Boundry, 4, 24,
qd.screenBits.bounds.right - 4,
qd.screenBits.bounds.bottom - 4);

DragWindow(WhichWindow, AnEvent.where, &Boundry);
break;
default:
break;
}
break;

default:
break;
}
}
}
}

void ArpClose(void)
{
if (GManualPatch)
{

```

```

SavePatch(timePortID, timePortResInfoID, "timePortInfo");

SavePatch(inputPortID, inputPortResInfoID,
"inputPortInfo");

SavePatch(outputPortID, outputPortResInfoID,
"outputPortInfo");
}
MIDISignOut(arpClientID);
}

void StopDialog(void)
{
StdSaveDLOGLocation(GMainDialog, mainDialogID);
HideWindow(GMainDialog);
DisposDialog(GMainDialog);
}

pascal short
ArpReader(MIDIPacket *ThePacketPtr, long TheRefCon)
{
long SysA5 = SetA5(TheRefCon);
short RetVal = midiMorePacket, i, j;

if (ArpGlobals.Locked)
{
RetVal = midiKeepPacket;
}
else if (ThePacketPtr->flags == stdPacketFlags
&& ((ThePacketPtr->data[0] & statusMask)
== keyOn || (ThePacketPtr->data[0]
& statusMask) == keyOff))
{
RetVal = midiMorePacket;
if ((ThePacketPtr->data[0] & statusMask)
== keyOn && ThePacketPtr->data[2] != zeroVelo)
{
for (i=0; i<ArpGlobals.NumNotes; i++)
{
if (ThePacketPtr->data[1] <=
ArpGlobals.NoteTbl[i].Note)
{
break;
}
}
for (j=ArpGlobals.NumNotes; j>i; j--)
{
ArpGlobals.
NoteTbl[j].Channel = ArpGlobals.NoteTbl[j-1].Channel;
ArpGlobals.
NoteTbl[j].Note = ArpGlobals.NoteTbl[j-1].Note;
ArpGlobals.
NoteTbl[j].Velocity = ArpGlobals.NoteTbl[j-1].
Velocity;
}
ArpGlobals.NoteTbl[i].Channel = ThePacketPtr->data[0]
& channelMask;
ArpGlobals.NoteTbl[i].Note = ThePacketPtr->data[1];
ArpGlobals.NoteTbl[i].Velocity = ThePacketPtr->data[2];

if (ArpGlobals.NumNotes < noteTblSize)
{
ArpGlobals.NumNotes++;
}
if (ArpGlobals.NumNotes == 1)
{
ArpGlobals.NextNoteOnTime = ThePacketPtr->tStamp;
ArpTimeProc(ThePacketPtr->tStamp, TheRefCon);
}
}
else if ((ThePacketPtr->data[0] &
statusMask) == keyOff || (ThePacketPtr->data[0] &
statusMask) == keyOn)
{

```



```

    for (i=0; i< ArpGlobals.NumNotes; i++)
    {
        if((ArpGlobals.NoteTbl[i].Channel ==
            (ThePacketPtr->data[0] & channelMask))
            && (ArpGlobals.NoteTbl[i].Note ==
                ThePacketPtr->data[1]))
        {
            break;
        }
    }

    if (i < ArpGlobals.NumNotes)
    {
        for (/*i=i*/; i<ArpGlobals.NumNotes-1; i++)
        {
            ArpGlobals.NoteTbl[i].Channel =
                ArpGlobals.NoteTbl[i+1].Channel;
            ArpGlobals.NoteTbl[i].Note =
                ArpGlobals.NoteTbl[i+1].Note;
            ArpGlobals.NoteTbl[i].Velocity =
                ArpGlobals.NoteTbl[i+1].Velocity;
        }
        ArpGlobals.NumNotes--;
    }
}

SetA5(SysA5);
return(RetVal);
}

pascal void ArpTimeProc(long , long TheRefCon)
{
    long SysA5 = SetA5(TheRefCon);
    int i;
    MIDIPacket TheMIDIPacket;

    ArpGlobals.Locked = 1;

    if (ArpGlobals.NumNotes == 0)
    {
        MIDIWakeUp(ArpGlobals.TimeRefNum, zeroTime,
            zeroPeriod, noTimeProc);
    }
    else
    {
        BumpNoteTableIndex();
        if (ArpGlobals.NoteTbl [ArpGlobals.NoteIndex]
            == ArpGlobals.LastNote)
        {
            BumpNoteTableIndex();
        }
        ArpGlobals.LastNote = ArpGlobals.NoteTbl[
            ArpGlobals.NoteIndex];

        TheMIDIPacket.flags = stdPacketFlags;
        TheMIDIPacket.len = keyOnOffPacketSize;

        i = ArpGlobals.NoteIndex;
        TheMIDIPacket.tStamp = ArpGlobals.NextNoteOnTime;
        TheMIDIPacket.data[0] = ArpGlobals.NoteTbl[i].
            Channel | keyOn;
        TheMIDIPacket.data[1] = ArpGlobals.NoteTbl[i].Note;
        TheMIDIPacket.data[2] = ArpGlobals.NoteTbl[i].Velocity;
        MIDIWritePacket(ArpGlobals.OutputRefNum, &TheMIDIPacket);

        TheMIDIPacket.tStamp += noteDuration;
        TheMIDIPacket.data[0] -= 0x10;
        MIDIWritePacket(ArpGlobals.OutputRefNum, &TheMIDIPacket);

        ArpGlobals.NextNoteOnTime += ArpGlobals.Tempo;

        MIDIWakeUp(ArpGlobals.TimeRefNum,

```

```

        ArpGlobals.NextNoteOnTime - ArpGlobals.Tempo/2,
        zeroPeriod, (ProcPtr) ArpTimeProc);
    }
    ArpGlobals.Locked = false;

    SetA5(SysA5);
}

void BumpNoteTableIndex(void)
{
    if (ArpGlobals.NumNotes == 1)
    {
        ArpGlobals.NoteIndex = 0;
    }
    else
    {
        switch(ArpGlobals.ArpPattern)
        {
            case patternUpID:
                if (ArpGlobals.NoteIndex >= ArpGlobals.
                    NumNotes-1)
                {
                    ArpGlobals.NoteIndex = 0;
                }
                else
                {
                    ArpGlobals.NoteIndex++;
                }
                break;

            case patternUpDownID:
            case patternDownUpID:
                if (ArpGlobals.ArpDirection == goingUp)
                {
                    if (ArpGlobals.NoteIndex >
                        ArpGlobals.NumNotes-1)
                    {
                        ArpGlobals.NoteIndex = ArpGlobals.NumNotes-1;
                        ArpGlobals.ArpDirection = goingDown;
                    }
                    else if (ArpGlobals.NoteIndex ==
                        ArpGlobals.NumNotes-1)
                    {
                        ArpGlobals.NoteIndex--;
                        ArpGlobals.ArpDirection = goingDown;
                    }
                    else
                    {
                        ArpGlobals.NoteIndex++;
                    }
                }
                else
                {
                    if (ArpGlobals.NoteIndex < 0)
                    {
                        ArpGlobals.NoteIndex = 0;
                        ArpGlobals.ArpDirection = goingUp;
                    }
                    else if (ArpGlobals.NoteIndex == 0)
                    {
                        ArpGlobals.NoteIndex++;
                        ArpGlobals.ArpDirection = goingUp;
                    }
                    else
                    {
                        ArpGlobals.NoteIndex--;
                    }
                }
                break;

            case patternDownID:
                if (ArpGlobals.NoteIndex <= 0)
                {

```

```

        ArpGlobals.NoteIndex = ArpGlobals.
            NumNotes-1;
    }
    else
    {
        ArpGlobals.NoteIndex--;
    }
    break;

case patternRandomID:
    ArpGlobals.NoteIndex
    = Choose(ArpGlobals.NumNotes);
    break;
}
}

void PatchPorts(void)
{
    MIDIPortInfoHdl PortInfoH;
    MIDIPortInfoPtr PortInfoP;
    short i, TheErr;

    PortInfoH = (MIDIPortInfoHdl)
        GetResource(portResType, timePortResInfoID);
    if (PortInfoH == NULL)
    {
        ReportResError("GetResource(portResType,
            timePortResInfoID)");
    }
    HLock((Handle) PortInfoH);
    PortInfoP = *PortInfoH;
    if (GetHandleSize((Handle) PortInfoH) != 0)
    {
        if (PortInfoP->timeBase.clientID != noClient)
        {
            TheErr = MIDICheckTime(PortInfoP->
                timeBase.clientID, PortInfoP->
                timeBase.portID, arpClientID, timePortID);
            if (TheErr != midiVConnectErr)
            {
                MIDISetSync(ArpGlobals.TimeRefNum,
                    midiExternalSync);
            }
        }

        for (i=0; i<PortInfoP->numConnects; i++)
        {
            MIDICheckTime(arpClientID, timePortID,
                PortInfoP->cList[i].clientID,
                PortInfoP->cList[i].portID);
        }
    }
    HUnlock((Handle) PortInfoH);
    ReleaseResource((Handle) PortInfoH);
    ReportResError("PatchPorts/ReleaseResource()");

    PortInfoH = (MIDIPortInfoHdl)
        GetResource(portResType, inputPortResInfoID);
    if (PortInfoH == NULL)
    {
        ReportResError("PatchPorts/GetResource()");
    }
    HLock((Handle) PortInfoH);
    PortInfoP = *PortInfoH;
    if (GetHandleSize((Handle) PortInfoH) != 0)
    {
        for (i=0; i<PortInfoP->numConnects; i++)
        {
            MIDICheckData(arpClientID, inputPortID,
                PortInfoP->cList[i].clientID,
                PortInfoP->cList[i].portID);
        }
    }
}

```

```

    }
    HUnlock((Handle) PortInfoH);
    ReleaseResource((Handle) PortInfoH);
    ReportResError("PatchPorts/GetResource()");

    PortInfoH = (MIDIPortInfoHdl)
        GetResource(portResType, outputPortResInfoID);
    if (PortInfoH == NULL)
    {
        ReportResError("PatchPorts/GetResource()");
    }
    HLock((Handle) PortInfoH);
    PortInfoP = *PortInfoH;
    if (GetHandleSize((Handle) PortInfoH) != 0)
    {
        for (i=0; i<PortInfoP->numConnects; i++)
        {
            MIDICheckData(arpClientID, outputPortID,
                PortInfoP->cList[i].clientID,
                PortInfoP->cList[i].portID);
        }
    }
    HUnlock((Handle) PortInfoH);
    ReleaseResource((Handle) PortInfoH);
    ReportResError("PatchPorts/ReleaseResource()");
}

void SavePatch(OSType PortID, short PortInfoResID, char
    *PortInfoResName)
{
    Handle PortResH;
    CursHandle WatchCurs;

    WatchCurs = GetCursor(watchCursor);
    HLock((Handle) WatchCurs);
    SetCursor(*WatchCurs);
    HUnlock((Handle) WatchCurs);

    PortResH = GetResource(portResType,
        PortInfoResID);
    ReportResError("SavePatch/GetResource()");
    RmveResource(PortResH);
    ReportResError("SavePatch/RmveResource()
        (make sure disk is unlocked)");
    DisposHandle(PortResH);
    UpdateResFile(CurResFile());
    ReportResError("SavePatch/UpdateResFile()
        (make sure disk is unlocked)");

    PortResH = (Handle) MIDIGetPortInfo(
        arpClientID, PortID);

    addressource(PortResH, portResType, PortInfoResID,
        PortInfoResName);
    ReportResError("SavePatch/addressource()");
    WriteResource(PortResH);
    ReportResError("SavePatch/WriteResource()
        (make sure disk is unlocked)");
    UpdateResFile(CurResFile());
    ReportResError("SavePatch/UpdateResFile()
        (make sure disk is unlocked)");
    ReleaseResource(PortResH);
    ReportResError("SavePatch/ReleaseResource()
        (make sure disk is unlocked)");
    InitCursor();
}

void Terminate(void)
{
    MIDISignOut(arpClientID);
    StopDialog();
    ExitToShell();
}

```



Advanced Mac'ing

Inside Controls



Ken Earle
Toronto, Canada
Volume 5, Number 11

[Ken Earle has been programming the Mac in C for about two years and still thinks it is fun. In between Mac contracts, he waits for the price of Finale to drop...]

If the Control Manager chapter of Inside Macintosh has you resignedly scrolling back to reread QuickDraw, here's a different point of view that might help. Or if scroll bars have lost their thrill and you're looking for a new toy, absorbing the code from this article will let you design your own custom controllable rotary-indicator digital-readout gizmos, guaranteeing hours of harmless entertainment.

First we'll take a look at implementing meters (aw, you guessed) in a way that's functionally very close to the Control Manager (see the listings for Meter.h and Meter.c). Then, after a strategy session on scroll bars, you'll be ready for MeterTest, an application with fully-functional scroll bars and windows that can serve as a design rig for meters or as a skeleton for something grander (see MeterMain.c).

If you're not using Think C, you'll need to at least #include the standard Mac header files, and check usage of the QuickDraw global 'thePort' (called 'qd.thePort' in MPW C).

The Meter Manager

I don't know about you, but if I'm given the choice between a detailed description of a routine and the source code for it, I'd rather have the source code. Especially for those darn scroll bars, where, as Ted Nelson might put it, everything gets deeply intertwined. To show how the Control Manager calls work on the inside and still end up with relatively useful code, let's take a shot at creating a cousin of the scroll bar, namely something like that nifty meter on page 312 of Inside Macintosh Volume 1. To better illustrate what the Control Manager does, the approach used is not to write a CDEF but rather to imitate most of the Control Manager routines, with Meters replacing Controls. The variations are minor enough that you should be able to get a good idea of what a "Control" routine does by looking at the corresponding "Meter" routine. For example, if you're interested in TrackControl(), take a look at TrackMeter() in the listing for Meter.c.

The routines in Meter.c do not duplicate all of the Control Manager's functions (but enough, I hope), and are virtually guaranteed not to resemble Apple's source code except in functionality since I didn't peek at it, honest. Meter.c is meant to be read with Inside Macintosh on your lap and should be otherwise self-explanatory, provided you introduce yourself to what a meter is first.

Prepare To Make Your Meter

Figure 1 shows a typical meter. It has a needle that wiggles

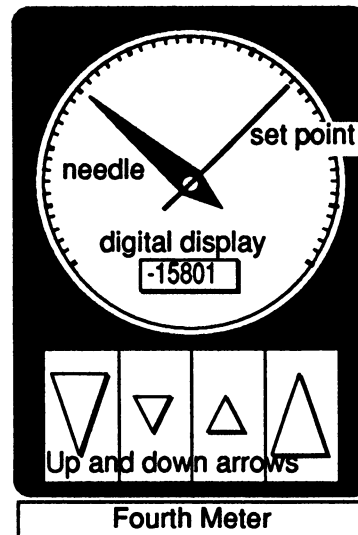


Figure 1. A Typical Meter

around in response to a continual input, a set point indicator that can be used for an alarm or for closed-loop control, four up and down "arrows" to move the set point around, a digital display, a title, and some background drawing to make it look like a meter. The most important difference between a meter and a scroll bar is that a meter doesn't have a true thumb—the set point indicator would be the natural candidate, but it is too small and slippery a target to be picked up and

dragged around, especially if it is used for something like controlling the temperature in a nuclear reactor. Instead, meters have a "background" part; a mouse down in anything except the up and down arrows will cause FindMeter() to return a part code for the background, and TrackMeter() will switch the digital display from the current input value to the set point value. For a look inside the scroll bar thumb see "Where's the thumb?" below.

Meters keep track of two values at once; the "meterValue", which corresponds to a regular "controlValue", represents the set point indicator which is under the user's control by means of the up and down arrows, and the "needleValue" is the current input to the meter and of course drives the ever-wandering needle. The needle has no analog (bad pun intended) in a regular control.

Meters can be any size above a reasonable minimum, but their proportions are fixed since they are delicate creatures to draw (the dial looks funny if it isn't round). You supply the top, left, and right for a new meter, and NewMeter() calculates the bottom for you. Many aspects of a meter's appearance can be modified by playing with the #defined constants at the top of the Meter.c listing, but for radical changes you'll want to dig into the details of CreateMeterBackground(), DrawNeedle(), etc. I like to think I've deliberately left lots of room for improvement....

Although you can treat meters mostly the same way as scroll bars in an application, there are two important differences. First, the meter needle usually needs to be driven constantly, so you'll probably want to call SetNeedleValue() at the top of your main event loop—you'll see an example of this in MeterMain.c. And

second, meters attach themselves as a linked list to their window's refCon field, in the same way that controls attach themselves to the window's controlList field. If you're already using this field to hold a reference to your own data structure, you should add a long int field "meterRef" to your data structure and replace the references in Meter.c to

```
(MeterHandle)(wPkt->refCon)
with something like
(MeterHandle)(myDataStructurePointer->meterRef).
```

Scroll Bars and Drawing

Scroll bars by themselves just keep track of the value of one integer, and show you a rough indication of how big it is relative to a maximum and minimum. What you do with that value is up to you. The code in MeterMain.c demonstrates in a general way how to use scroll bars for controlling the position of fixed-size documents that contain graphics within fully-functional windows. Meters are used for the graphics, and when you build all of the listings into the application MeterTest you'll also have a tool for designing meters (see About MeterTest below).

The key to tying scroll bars, windows, and drawing together is to introduce another system of coordinates (as if local and global weren't enough). Think of the graphics as being attached beneath the window to a piece of paper which we'll call the document. "Document" coordinates are for that piece of paper, running from 0,0 at left top down to 'page width', 'page height' at right bottom, and coincide with unscrolled coordinates when no scrolling has taken place. These numbers are usually in pixels, since they then describe positions in the grafPort directly.

Scroll bars are used to relate local and document coordinates in the natural way that comes about if you think of dropping the window on top of the document. The scroll bar minimums are set to zero, corresponding to the top left of the document. The scroll bar values record the horizontal and vertical coordinates of the top left of the window in document coordinates (which are always at least as large as local coordinates). The scroll bar maximums are equal to the amount of the document left unseen, so the vertical maximum for example is equal to Document Height less the PortRect Height less the 15 pixels covered by the horizontal scroll bar. To put it another way, see Figure 2.

The overall strategy is to always draw in document coordinates by taking care of scrolling before the low-level drawing routines are called. Done properly, your drawing routines can pretend that they are always drawing to a giant screen that shows the whole (unscrolled) document at once. "Taking care of scrolling" is very easy. Before doing any drawing, shift the whole window into document coordinates by means of SetOrigin(hBarValue, vBarValue), and shift the clipping region too with OffsetRgn(thePort->clipRgn, hBarValue, vBarValue)—then shift back when you're done. When you're deciphering where the mouse is, local points from GetMouse() can be translated to document points by adding the scroll bar values to the appropriate coordinates (see DocToLocal()).

Your drawing routines will draw to the right place if they take a point or rectangle in document coordinates to start from, and then draw relative to the point or rectangle without referring

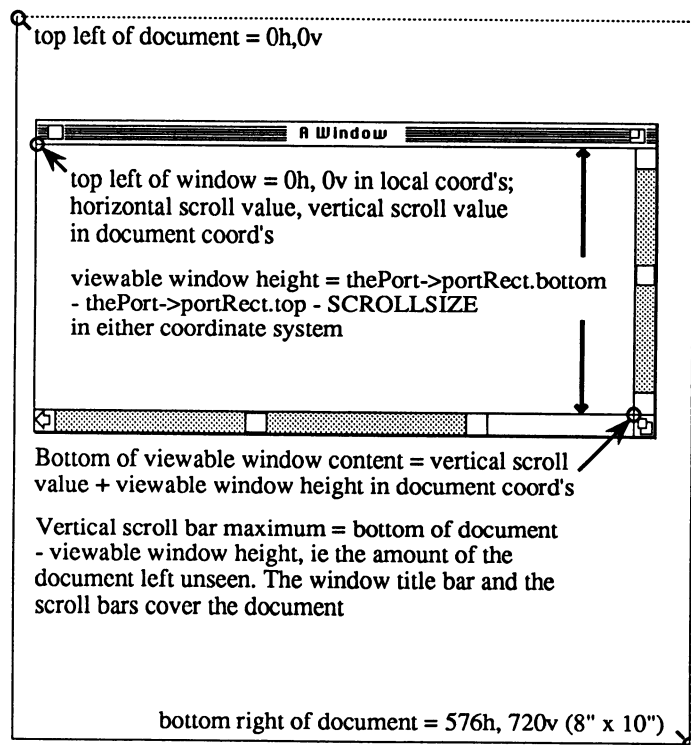


Figure 2. Scroll Bar Values

to the scroll bars. For example, the body of a function to draw a straight horizontal line could look like

```
(MoveTo(docPoint.h, docPoint.v);
Line(widthInPixels, 0);)
However, something like
MoveTo(docPoint.h + hBarValue, docPoint.v + vBarValue);
```

probably won't do what you intended, since it is trying to shift from local to document coordinates, but that's already been done beforehand. If you need to refer to the window frame, the left edge of the window is still at thePort->portRect.left on the screen, but remember that this value isn't necessarily zero.

Drawing needs to be done whenever the user wants something new (eg mouse down with a "pencil" tool, or <Command>-drag to resize a meter in MeterTest), and when events cause a hidden part of the document to be exposed (eg activating a different window, or scrolling). This redrawing as opposed to original drawing can all be handled by one routine, and this same routine can even draw to the clipboard or printer if you plan ahead. The strategy here is to: determine what part of the window needs redrawing; shift to document coordinates; and draw only those objects that have been newly exposed by calling the low-level drawing routines for those objects. UpdateContents() is the redrawing function in MeterMain.c.

The easiest way to keep track of what part of the window needs redrawing is to use a global region and set this region to the appropriate area just before calling the redraw function. If you glance through MeterMain.c you'll see that the region updateRgn is always set just before calling UpdateContents(). In principle the updateRgn can always be figured out exactly (see eg DoGrowWindow() and ScrollProc()), but sometimes it is ac-

ceptably fast to just cop out by invalidating the whole window and letting the generated update event take care of the drawing (see DoZoomWindow(), and "case updateEvt" in main()).

I first ran into the idea of using document coordinates and a global update region in "Hidden Powers of the Macintosh" by Christopher Morgan (The Waite Group), still one of the best introductions to programming the Mac in spite of all the code being in Pascal.

The redrawing for the arrow and paging parts of scroll bars is done in MeterMain.c by the function ScrollProc(), where both the moving of the screen image and the accumulation of the newly exposed region are managed by a single call to ScrollRect(). You could if you prefer handle paging by invalidating the whole window (less scroll bars) and calling UpdateContents()—this is faster, but less visually appealing.

That just leaves the scroll bar thumb. I could have made it real easy for you, but some people just can't leave well enough alone....

Where's the thumb?

If you call TrackControl() with no action procedure in response to a mouse down in a scroll bar thumb, everything is taken care of for you. After the call, you just redraw the window if the new control setting differs from the old one. You can even get fancy and call ScrollRect() and UpdateContents() if the move was a small one, or invalidate the whole window if it was a biggy.

One thing you can't do this way, however, is keep track of where the thumb is while the user is dragging it to and fro. In order to do this you have to know the "potential" new control value for each new mouse position, but not even TrackControl() bothers to figure this out. It just drags an outline of the thumb around without regard for value, and only calculates a new value when the mouse button is released. You could write an action procedure for TrackControl() that figures out where the thumb is, but in the spirit of getting inside things I have written a replacement for TrackControl() to be called for the thumb. ThumbControl() in MeterMain.c does all that TrackControl() does for the thumb, and also calculates the potential new control value for you (as well as the thumb's physical position). You can insert your custom routine calls at the places marked in the code, or modify the function to accept pointers to functions if you can't stand inelegance.

While you could use this approach to place a page number directly inside the thumb, the approach used by WriteNow for example of putting the page number alongside a scroll bar is simpler and also better—try putting a four-digit page number inside the scroll box! MeterMain.c by the way shows how to put a numeric display next to a scroll bar, something that's especially useful when designing or debugging.

About MeterTest

If you build an application from the files in this article according to the instructions at the top of the listing for MeterMain.c you'll have MeterTest, which sums everything up by presenting meters as graphic objects inside of windows with scroll bars. The underlying documents which contain the meters

are fixed in size (8" by 10"), and all the usual scrolling and window functions are implemented in a general way except for the lowest-level drawing routines.

MeterTest can also serve as a test rig for modifying the appearance and function of meters. You'll have use an edit-and-recompile approach, but fortunately MeterTest is small and Think C is fast. Many changes can be made by altering the #defines at the top of Meter.c, but others will require changing the routines that create the meter's background and draw the other parts. The titles for the four default meters and two windows are at the top of MeterMain.c, and other standard parameters for the meters are set inside the function MakeTwoMeters().

To move a meter around, Option-drag it. Command-drag on a meter to resize it constantly while the button is down. The display box at lower right of the window shows the horizontal and vertical scroll bar values unless you resize a meter, in which case it shows the meter width and height. Click on an up or down triangle in a meter to move the set point indicator up and down, or click anywhere else on a meter to change the meter's display from the current needle input to the set point value.

Bonus round

In the accompanying code you'll also find examples of simple animation, using integer tables to represent transcendental functions, ring-shaped clipping, how to shadow any polygon, handling linked lists and pointers to functions...and after you've worked through it all, you'll never again be intimidated by a stupid thumb.

Listing: Meter.h

```
/*Meter.h - MeterRecord definition & interface function
prototypes*/

/* special partCode for meter */
#define inBackground 24

struct MeterRecord
(
    struct MeterRecord **nextMeter; /* or NULL */
    WindowPtr meterOwner;
    Rect meterRect; /* includes title*/
    Byte meterVis;
    Byte meterHilite;
    long meterValue, meterMin, meterMax,
        needleValue;
    /*ProcPtr meterAction; not implemented */
    long meterRfCon;
    char title[32]; /* pascal format, length byte first */
    /* some additional fields not in a control record */
    Rect incDownRect, incUpRect,
        pageDownRect, pageUpRect;
    Rect displayRect;
    long displayDivisor; /* power of ten */
    int needleShort, needleLong, halfNeedleWidth;
    int setShort, setLong, halfSetWidth;
    PicHandle backgroundH;
    PolyHandle incUpH, incDownH, pageUpH, pageDownH;
    Point dialCentre;
    long hasMeterID;
);

typedef struct MeterRecord MeterRecord;
typedef MeterRecord *MeterPtr;
typedef MeterRecord **MeterHandle;
```

```
typedef void (*voidPtr)();

/* Functions defined in Meter.c */
extern MeterHandle NewMeter(WindowPtr, Rect*, char*,
    Boolean, long, long, long);
extern void DisposeMeter(MeterHandle);
extern void KillMeters(WindowPtr);
extern void SetMTitle(MeterHandle, char*);
extern void GetMTitle(MeterHandle, char*);
extern void HideMeter(MeterHandle);
extern void ShowMeter(MeterHandle);
extern void DrawMeters(WindowPtr);
extern void HiliteMeter(MeterHandle, int);
extern int FindMeter(Point, WindowPtr, MeterHandle*);
extern int TrackMeter(MeterHandle, Point, voidPtr);
extern int TestMeter(MeterHandle, Point);
extern void MoveMeter(MeterHandle, int, int);
extern void DragMeter(MeterHandle, Point, Rect*, Rect*, int);
extern void SizeMeter(MeterHandle, int);
/* note proportions are fixed */
extern void SetMtrValue(MeterHandle, long);
extern long GetMtrValue(MeterHandle);
extern void SetMtrMin(MeterHandle, long);
extern long GetMtrMin(MeterHandle);
extern void SetMtrMax(MeterHandle, long);
extern long GetMtrMax(MeterHandle);
extern void SetMRefCon(MeterHandle, long);
extern long GetMRefCon(MeterHandle);
/* note multiple actions procs not supported in this version */
extern void SetNeedleValue(MeterHandle, long);
extern long GetNeedleValue(MeterHandle);
extern long GetDisplayDivisor(MeterHandle);
extern Boolean ValidMeter(MeterHandle);
extern int GetMaxTitlePixels(MeterHandle);
extern void MeterSnapshot(MeterHandle);
/* draws to thePort rather than meter's owning window */
```

Listing: Meter.c

```
/* Meter.c - display, drive & control a meter with set point.
Usage: add 'Meter.c' to your project, and #include "Meter.h"
in any file that refers to the functions below. Add 'Math' and
#include <math.h> if you don't hand-load the sineTable[].
```

- the main purpose here is to show how the Control Manager works, so we'll use functional equivalents of most of the Control Manager routines listed in Inside Macintosh vol 1 pages 309 - 338 rather than creating a new kind of control. In general, "Control" in IM is replaced by "Meter" here; for example, if you're interested in how NewControl() works see NewMeter() below. See MeterProc() in MeterMain.c for an action procedure that is called by TrackMeter() to drive the set point indicator up and down. Fancy stuff requiring external user-defined resources is not included (no GetNewMeter()), nor is a default action Proc supported (always call TrackMeter() with your action Proc or 0L). Two other differences that no one should mind - all functions are 'C' rather than 'Pascal', and all value parameters are long.
- meterValue refers to the set point value, which is normally under user control, and can be set by SetMtrValue().
- needleValue refers to current input to the meter, displayed by the needle and NOT normally under user's control, set by SetNeedleValue().
- the digital display normally shows needleValue unless mouse is down in meter background (simple equiv of control "thumb").
- for a more detailed look at controlling what happens when mouse is in an indicator see ThumbControl() in MeterMain.c.
- the arrow and paging parts appear as small and large triangles (defined as polygons), with the standard part codes.
- meter handles are linked to the owning windows' refCon field. If you want to use this field for something else, change the refCon references below in NewMeter(), DisposeMeter(), KillMeters(), DrawMeters(), and FindMeter().

```
*/
```

```
/* Do not include Meter.h, but DO #include <MacHeaders> either
explicitly here or by setting the "MacHeaders" option */
```

```
/* math.h is needed for sineTable[], otherwise hand-load it.*/
#define _ERRORCHECK_
int errno;
#include <math.h>

/* special partCode for meter, everything but the four up and
down arrows. A meter has no true thumb - the set point
indicator is a skinny rotating target, and too dangerous to
grab if the meter is being used for closed-loop control.
"Track and hilite the background" just means show the set
point value in the display rather than the current needle
value while the mouse is down.*/
#define inBackground 24

#define NULL 0L
#define HASMETER 'qwer' /* arbitrary long ID,
    used by ValidMeter() */

/* Some #defines to vary the appearance of the meter. Since
meters are complicated beasts to draw, be prepared to fine-
tune the drawing code as well if you explore different looks!
*/
#define MINWIDTH 48 /* min. usable meter width */
#define TVHRATIO 1400L
/* black rounded rect height/width, times 1000 */
#define TITLEHEIGHT 24 /* half height of title
    box below meter proper, plus 4 */
/* given width of meter = w, total height is (w*TVHRATIO)/1000
+ TITLEHEIGHT */
#define TEXTVOFFSET 4 /* to centre 12 pt
    text in the display box */
#define INNERPCT 94L /* used for defining needle
    and set point lengths as a % of dial radius */
#define HALFSETWIDTH 1 /* half width of set
    point indicator */
#define CENTRERADIUS 3 /* radius of meter's
    little centre circle */
#define PGPOLYHEIGHT 3000L
/* times 1000, keep it between 1000 and 3000 */
#define INCPOLYHEIGHT 2000L
/* -POLYHEIGHT is ratio of height to width of set point
control triangles times 1000 - try 1000, 1414, 1732, 2000, 3000. */
#define NUMTICKS 11 /* number of major position ticks
    around circumference */
#define NUMMINORTICKS 51 /* NUMMINORTICKS includes
    NUMTICKS.*/
#define HALFNEEDLERATIO 8 /* dial radius/
    HALFNEEDLERATIO = halfNeedleWidth */
#define MINDIGITS 3 /* minimum number of
    digits to display */
#define MAXDIGITS 5 /* maximum number of
    digits to display */

/* The digital display will show at least MINDIGITS for small
meters, and at most MAXDIGITS for large meters. See
ShowValue(), AdjustDisplay(), and SetDisplayDivisor() for
details. As a nicety, you should update the meter title to
reflect the actual units displayed (eg if input is Volts and
displayDivisor is 10, the meter name should be something like
"Volts / 10" */
```

```
static int sineTable[91]; /* set up by InitMeters() */
```

```
struct MeterRecord
{
    struct MeterRecord **nextMeter; /* or NULL */
    WindowPtr meterOwner;
    Rect meterRect; /* includes title */
    Byte meterVis;
    Byte meterHilite;
    long meterValue, meterMin,
```

```

        meterMax, needleValue;
/*ProcPtr meterAction; - not implemented */
long    meterRfCon;
char    title[32]; /* pascal format, length byte first */
/* some additional fields not in a control record */
Rect    incDownRect, incUpRect,
        pageDownRect, pageUpRect;
Rect    displayRect; /* digital display */
long    displayDivisor; /* power of ten */
int     needleShort, needleLong, halfNeedleWidth;
int     setShort, setLong, halfSetWidth;
PichHandle backgroundH;
PolyHandle incUpH, incDownH, pageUpH, pageDownH;
/* triangles */
Point   dialCentre;
long    hasMeterID; /* = HASMETER for a meter */
};

typedef struct MeterRecord MeterRecord;
typedef MeterRecord *MeterPtr;
typedef MeterRecord **MeterHandle;

typedef void (*voidPtr)(); /*true C equiv. of procPtr */

/* Functions defined in this file */

/* External interface, mostly like Control Manager */
/* Control Manager equivalents */
MeterHandleNewMeter(WindowPtr, Rect*, char*,
                    Boolean, long, long, long, long);
void DisposeMeter(MeterHandle);
void KillMeters(WindowPtr);
void SetMTitle(MeterHandle, char*);
void GetMTitle(MeterHandle, char*);
void HideMeter(MeterHandle);
void ShowMeter(MeterHandle);
void DrawMeters(WindowPtr);
void HiliteMeter(MeterHandle, int);
int FindMeter(Point, WindowPtr, MeterHandle*);
int TrackMeter(MeterHandle, Point, voidPtr);
int TestMeter(MeterHandle, Point);
void MoveMeter(MeterHandle, int, int);
void DragMeter(MeterHandle, Point, Rect*, Rect*, int);
void SizeMeter(MeterHandle, int); /* note
                                proportions are fixed */
void SetMtrValue(MeterHandle, long); /* use
                                this to move the set point */
long GetMtrValue(MeterHandle);
void SetMtrMin(MeterHandle, long);
long GetMtrMin(MeterHandle);
void SetMtrMax(MeterHandle, long);
long GetMtrMax(MeterHandle);
void SetMtrRfCon(MeterHandle, long);
long GetMtrRfCon(MeterHandle);
/* Special meter functions */
void SetNeedleValue(MeterHandle, long); /* use this
                                to move the needle */
long GetNeedleValue(MeterHandle);
long GetDisplayDivisor(MeterHandle);
Boolean ValidMeter(MeterHandle);
int GetMaxTitlePixels(MeterHandle); /* displayable
                                title width */
void MeterSnapshot(MeterHandle); /* draws meter
                                to thePort rather than owning window */

/* support functions used in this file only */
void InitMeters(void);
void CreateMeterBackground(MeterHandle);
void DrawMeterBackground(MeterHandle);
void DrawMeterTitle(MeterHandle);
void DrawTriangle(MeterHandle, int);
void DrawNeedle(MeterHandle);
void DrawSet(MeterHandle);
void ShowValue(MeterHandle, Boolean);
void DrawInactiveMeter(MeterHandle);

```

```

void AdjustDisplay(MeterHandle);
void SetDisplayDivisor(MeterHandle);
void GetSinAndCos(int, long*, long*);

/* New meter handles are linked to the window's refCon field.
Meter proportions are fixed; height is determined from width,
and boundRect.bottom is ignored. There is no default action
procedure. Titles are truncated on the right.*/
MeterHandleNewMeter(wPtr, boundRect, title, visible,
                    meterValue, meterMin, meterMax, meterRfCon)
WindowPtr wPtr;
Rect *boundRect;
char *title;
Boolean visible;
long meterValue, meterMin, meterMax;
long meterRfCon;
{
    int i, j;
    MeterHandle mH, tempMH;
    MeterPtr mPtr;
    WindowPeek wPk = (WindowPeek)wPtr;
    char *titlePtr;

    /* Initialize sine table if necessary. */
    if (sineTable[45] == 0) /* should be 707 */
        InitMeters();
    /* Allocate and attach a new meter record. */
    mH = (MeterHandle)NewHandle(sizeof(MeterRecord));
    if (MemError() != noErr)
        return(NULL);
    mPtr = *mH;
    mPtr->meterOwner = wPtr;
    mPtr->nextMeter = NULL;
    mPtr->hasMeterID = HASMETER;
    /* refCon of WindowRecord holds first MeterHandle. New
    meters are inserted at beginning of list. */
    tempMH = (MeterHandle)(wPk->refCon);
    wPk->refCon = (long)mH;
    if (tempMH && ((*tempMH).hasMeterID == HASMETER))
        mPtr->nextMeter = tempMH;
    /* Fill in the easy fields. */
    mPtr->meterVis = visible;
    mPtr->meterHilite = 0; /* active */
    /* meterMin <= meterValue <= meterMax */
    if (meterMax < meterMin)
    {
        DisposeMeter(mH);
        return(NULL);
    }
    if (meterValue < meterMin)
        meterValue = meterMin;
    if (meterValue > meterMax)
        meterValue = meterMax;
    if (meterMin == meterMax)
        (**mH).meterHilite = 255; /*inactive-see IM 1-327 */
    mPtr->meterValue = meterValue;
    mPtr->meterMin = meterMin;
    mPtr->meterMax = meterMax;
    mPtr->meterRfCon = meterRfCon;
    mPtr->needleValue = meterMin; /* default value */
    j = title[0];
    if (j > 31)
        j = 31;
    titlePtr = mPtr->title;
    for (i = 0; i <= j; ++i)
        *titlePtr++ = *title++;
    (**mH).meterRect = *boundRect;
    (**mH).halfSetWidth = HALFSETWIDTH;
    /* Do the grunt work. */
    CreateMeterBackground(mH);
    /* Draw meter if visible. */
    if (visible)
    {
        (**mH).meterVis = FALSE; /* forces drawing */
    }
}

```

```

    ShowMeter(mH);
}
return(mH);
} /* end NewMeter() */

void DisposeMeter(mH)
MeterHandle mH;
{
    Rect    tempRect;
    PicHandle tempPH;
    PolyHandle tempPoly;
    MeterHandle tMH;
    GrafPtr    savePort;

    if (!ValidMeter(mH)) return;
    GetPort(&savePort);
    SetPort((**mH).meterOwner);
    tempPH = (**mH).backgroundH;
    KillPicture(tempPH);
    tempPoly = (**mH).incUpH;
    KillPoly(tempPoly);
    tempPoly = (**mH).incDownH;
    KillPoly(tempPoly);
    tempPoly = (**mH).pageUpH;
    KillPoly(tempPoly);
    tempPoly = (**mH).pageDownH;
    KillPoly(tempPoly);
    tempRect = (**mH).meterRect;
    EraseRect(&tempRect);
    InvalRect(&tempRect);
    /* Unlink the meter. */
    tMH = (MeterHandle)
    (((WindowPeek)((**mH).meterOwner))>refCon);
    if (tMH != mH)
    {
        while ((**tMH).nextMeter != mH)
            tMH = (**tMH).nextMeter;
        (**tMH).nextMeter = (**mH).nextMeter;
    }
    else
        ((WindowPeek)((**mH).meterOwner))>refCon =
    (long)((**mH).nextMeter);
    DisposHandle((Handle)mH);
    SetPort(savePort);
} /* end DisposeMeter() */

void KillMeters(wPtr)
WindowPtr wPtr;
{
    WindowPeek wPk = (WindowPeek)wPtr;

    while (wPk->refCon && (((**((MeterHandle)
    (wPk->refCon)))>hasMeterID == HASMETER))
        DisposeMeter((MeterHandle)(wPk->refCon));
} /* end KillMeters() */

void SetMTitle(mH, title)
MeterHandle mH;
char *title;
{
    int j, i = title[0];
    GrafPtr savePort;
    Rect tempRect;

    if (!ValidMeter(mH)) return;
    if (i > 31)
        i = 31;
    if (i)
    {
        for (j = 0; j <= i; ++j)
            (**mH).title[j] = title[j];
        GetPort(&savePort);
        SetPort((**mH).meterOwner);
        DrawMeterTitle(mH);
    }
}

```

```

    SetPort(savePort);
}
} /* end SetMTitle() */

void GetMTitle(mH, title)
MeterHandle mH;
char *title;
{
    int j, i = (**mH).title[0];
    if (!ValidMeter(mH))
    {
        title[0] = 0;
        return;
    }
    for (j = 0; j <= i; ++j)
        title[j] = (**mH).title[j];
} /* end GetMTitle() */

void HideMeter(mH)
MeterHandle mH;
{
    Rect tempRect;
    GrafPtr savePort;

    if (!ValidMeter(mH)) return;
    if (!(**mH).meterVis) return;
    GetPort(&savePort);
    SetPort((**mH).meterOwner);
    (**mH).meterVis = FALSE;
    tempRect = (**mH).meterRect;
    EraseRect(&tempRect);
    InvalRect(&tempRect);
    SetPort(savePort);
} /* end HideMeter() */

void ShowMeter(mH)
MeterHandle mH;
{
    int i;
    GrafPtr savePort;

    if (!ValidMeter(mH)) return;
    if ((**mH).meterVis) return;
    GetPort(&savePort);
    SetPort((**mH).meterOwner);
    (**mH).meterVis = TRUE;
    DrawMeterBackground(mH);
    for (i = inUpButton; i <= inPageDown; ++i)
        DrawTriangle(mH, i);
    DrawNeedle(mH);
    DrawSet(mH);
    ShowValue(mH, TRUE);
    SetPort(savePort);
} /* end ShowMeter() */

void DrawMeters(wPtr)
WindowPtr wPtr;
{
    int i;
    GrafPtr savePort;
    MeterHandle mH = (MeterHandle)((WindowPeek)wPtr->refCon);

    if (!ValidMeter(mH)) return;
    GetPort(&savePort);
    SetPort(wPtr);
    while (mH)
    {
        if ((**mH).meterVis)
        {
            DrawMeterBackground(mH);
            for (i = inUpButton; i <= inPageDown; ++i)
                DrawTriangle(mH, i);
            DrawNeedle(mH);
            DrawSet(mH);
        }
        mH = (**mH).nextMeter;
    }
}

```



```

        ShowValue(mH, TRUE);
    }
    mH = (**mH).nextMeter;
}
SetPort(savePort);
} /* end DrawMeters() */

void HiliteMeter(mH, hiliteState)
MeterHandle mH;
int hiliteState;
{
    int i;
    GrafPtr savePort;

    if (!ValidMeter(mH)) return;
    if ((**mH).meterHilite == hiliteState) return;
    if ((**mH).meterHilite == 255 && hiliteState != 0) return;
    GetPort(&savePort);
    SetPort((**mH).meterOwner);
    if (inUpButton <= hiliteState &&
        hiliteState <= inPageDown)
    {
        (**mH).meterHilite = hiliteState;
        DrawTriangle(mH, hiliteState);
    }
    else if (hiliteState == inBackground)
    { /* see #define at top of file */
        (**mH).meterHilite = hiliteState;
        ShowValue(mH, FALSE);
    }
    else if (hiliteState == 0) /* make normal */
    {
        if ((**mH).meterHilite == 255)
        {
            (**mH).meterHilite = 0;
            DrawMeterBackground(mH);
            for (i = inUpButton; i <= inPageDown; ++i)
                DrawTriangle(mH, i);
            DrawNeedle(mH);
            DrawSet(mH);
            ShowValue(mH, TRUE);
        }
        else if (inUpButton <= (**mH).meterHilite &&
            (**mH).meterHilite <= inPageDown)
        {
            hiliteState = (**mH).meterHilite;
            (**mH).meterHilite = 0;
            DrawTriangle(mH, hiliteState);
        }
        else if ((**mH).meterHilite == inBackground)
        {
            (**mH).meterHilite = 0;
            ShowValue(mH, TRUE);
        }
    }
    else if (hiliteState == 255) /*make inactive*/
    {
        (**mH).meterHilite = hiliteState;
        DrawMeterBackground(mH);
    }
    SetPort(savePort);
} /* end HiliteMeter() */

int FindMeter(thePoint, wPtr, mHP)
Point thePoint;
WindowPtr wPtr;
MeterHandle *mHP;
{
    int partCode;
    Boolean foundIt = FALSE;
    MeterHandle mH = (MeterHandle)((WindowPeek)wPtr->refCon);

    while (ValidMeter(mH))
    {

```

```

        if (PtInRect(thePoint, &(**mH).meterRect))
        {
            foundIt = TRUE;
            break;
        }
        mH = (**mH).nextMeter;
    }
    if (foundIt && (partCode = TestMeter(mH, thePoint)))
    {
        *mHP = mH;
        return(partCode);
    }
    *mHP = NULL;
    return(0);
} /* end FindMeter() */

/* See ThumbControl() in MeterMain.c for an inside look at
tracking a moving indicator. */
int TrackMeter(mH, startPoint, actionProc)
MeterHandle mH;
Point startPoint;
voidPtr actionProc;
{
    int partCode;
    Point theMouse;
    Rect partRect, slopRect;
    Boolean pausing = FALSE;

    partCode = TestMeter(mH, startPoint);
    if (!partCode) return(0);
    if (inUpButton <= partCode &&
        partCode <= inPageDown)
    {
        /* Hilite part & call actionProc while mouse down in part. */
        if (partCode == inUpButton)
            partRect = (**mH).incUpRect;
        else if (partCode == inDownButton)
            partRect = (**mH).incDownRect;
        else if (partCode == inPageUp)
            partRect = (**mH).pageUpRect;
        else if (partCode == inPageDown)
            partRect = (**mH).pageDownRect;
        slopRect = partRect;
        InsetRect(&slopRect, -10, -10);
        HiliteMeter(mH, partCode);
        while (StillDown())
        {
            GetMouse(&theMouse);
            if (PtInRect(theMouse, &slopRect))
            {
                if (pausing)
                {
                    pausing = FALSE;
                    HiliteMeter(mH, partCode);
                }
                if (actionProc)
                    (*actionProc)(mH, partCode);
            }
            else
            {
                if (!pausing)
                {
                    pausing = TRUE;
                    HiliteMeter(mH, 0);
                }
            }
        }
        if (!pausing)
            HiliteMeter(mH, 0);
    }
    else if (partCode == inBackground)
    { /* see #define at top of listing */
        ShowValue(mH, FALSE);
        while (StillDown())

```

```

    {
        if (actionProc)
            (*actionProc)();
    }
    ShowValue(mH, TRUE);
}
else return(0);
} /* end TrackMeter() */

int TestMeter(mH, thePoint)
MeterHandle mH;
Point thePoint;
{
    int partCode;

    if (!ValidMeter(mH)) return(0);
    if (PtInRect(thePoint, &(**mH).meterRect))
    {
        if ((**mH).meterVis && ((**mH).meterHilite != 255))
        {
            if (PtInRect(thePoint, &(**mH).incUpRect))
                partCode = inUpButton;
            else if (PtInRect(thePoint, &(**mH).incDownRect))
                partCode = inDownButton;
            else if (PtInRect(thePoint, &(**mH).pageUpRect))
                partCode = inPageUp;
            else if (PtInRect(thePoint, &(**mH).pageDownRect))
                partCode = inPageDown;
            else
                partCode = inBackground;
            return(partCode);
        }
    }
    return(0);
} /* end TestMeter() */

void MoveMeter(mH, h, v)
MeterHandle mH;
int h, v;
{
    Boolean wasVisible = (**mH).meterVis;

    if (!ValidMeter(mH)) return;
    if (wasVisible)
        HideMeter(mH);
    OffsetRect(&(**mH).meterRect, h, v);
    OffsetRect(&(**mH).incDownRect, h, v);
    OffsetRect(&(**mH).incUpRect, h, v);
    OffsetRect(&(**mH).pageDownRect, h, v);
    OffsetRect(&(**mH).pageUpRect, h, v);
    OffsetRect(&(**mH).displayRect, h, v);
    (**mH).dialCentre.h += h;
    (**mH).dialCentre.v += v;
    if (wasVisible)
        ShowMeter(mH);
} /* end MoveMeter() */

void DragMeter(mH, startPoint, limitRect, slopRect, axis)
MeterHandle mH;
Point startPoint;
Rect *limitRect;
Rect *slopRect;
int axis;
{
    long posOffset;
    int h, v;
    Rect mRect, tRect;
    GrafPtr savePort;
    RgnHandle tempRgn;

    if (!ValidMeter(mH)) return;
    GetPort(&savePort);
    SetPort((**mH).meterOwner);
    tempRgn = NewRgn();

```

```

    mRect = (**mH).meterRect;
    mRect.bottom -= TITLEHEIGHT;
    tRect.left = (**mH).meterRect.left + 2;
    tRect.right = (**mH).meterRect.right - 2;
    tRect.bottom = (**mH).meterRect.bottom - 2;
    tRect.top = tRect.bottom - TITLEHEIGHT + 4;

    OpenRgn();
    FrameRoundRect(&mRect, mRect.right/10, mRect.bottom/10);
    FrameRect(&tRect);
    CloseRgn(tempRgn);

    posOffset = DragGrayRgn(tempRgn, startPoint,
                            limitRect, slopRect, axis, 0L);

    DisposeRgn(tempRgn);
    v = HiWord(posOffset);
    h = LoWord(posOffset);
    if (v != -32768 && h != -32768)
        MoveMeter(mH, h, v);
    SetPort(savePort);
} /* end DragMeter() */

/* Meter proportions are fixed - width determines height. */
void SizeMeter(mH, w)
MeterHandle mH;
int w;
{
    PichHandle tempPH;
    PolyHandle tempPoly;
    Boolean wasVisible = (**mH).meterVis;

    if (!ValidMeter(mH)) return;
    if (wasVisible)
        HideMeter(mH);
    /* gut the meter and recreate it */
    tempPH = (**mH).backgroundH;
    KillPicture(tempPH);
    tempPoly = (**mH).incUpH;
    KillPoly(tempPoly);
    tempPoly = (**mH).incDownH;
    KillPoly(tempPoly);
    tempPoly = (**mH).pageUpH;
    KillPoly(tempPoly);
    tempPoly = (**mH).pageDownH;
    KillPoly(tempPoly);
    tempPoly = (**mH).meterRect;
    KillPoly(tempPoly);
    (**mH).meterRect.right = (**mH).meterRect.left + w;
    CreateMeterBackground(mH);
    if (wasVisible)
        ShowMeter(mH);
} /* end SizeMeter() */

void SetMtrValue(mH, meterValue)
MeterHandle mH;
long meterValue;
{
    GrafPtr savePort;

    if (!ValidMeter(mH)) return;
    GetPort(&savePort);
    SetPort((**mH).meterOwner);
    DrawSet(mH); /* erases old pointer */
    /* pin value to range min:max */
    if (meterValue < (**mH).meterMin)
        meterValue = (**mH).meterMin;
    else if (meterValue > (**mH).meterMax)
        meterValue = (**mH).meterMax;
    (**mH).meterValue = meterValue;
    DrawSet(mH);
    ShowValue(mH, FALSE);
    SetPort(savePort);
} /* end SetMtrValue() */

long GetMtrValue(mH)
MeterHandle mH;

```

```

{
    if (!ValidMeter(mH)) return(0L);
    return(**mH).meterValue;
} /* end GetMtrValue() */

void SetMtrMin(mH, minValue)
MeterHandle mH;
long minValue;
{
    GrafPtr savePort;

    if (!ValidMeter(mH)) return;
    GetPort(&savePort);
    SetPort(**mH).meterOwner);
    DrawSet(mH); /* erases old pointer */
    DrawNeedle(mH); /* ditto needle */
    if (minValue >= (**mH).meterMax)
    {
        (**mH).meterMin = (**mH).meterMax;
        HiliteMeter(mH, 255); /* inactive - see IM 1-327 */
        return;
    }
    (**mH).meterMin = minValue;
    if ((**mH).meterValue < minValue)
        (**mH).meterValue = minValue;
    SetDisplayDivisor(mH);
    DrawSet(mH);
    DrawNeedle(mH);
    ShowValue(mH, FALSE);
    SetPort(savePort);
} /* end SetMtrMin() */

long GetMtrMin(mH)
MeterHandle mH;
{
    if (!ValidMeter(mH)) return(0L);
    return(**mH).meterMin;
} /* end GetMtrMin() */

void SetMtrMax(mH, maxValue)
MeterHandle mH;
long maxValue;
{
    GrafPtr savePort;

    if (!ValidMeter(mH)) return;
    GetPort(&savePort);
    SetPort(**mH).meterOwner);
    DrawSet(mH); /* erases old pointer */
    DrawNeedle(mH); /* ditto needle */
    if (maxValue <= (**mH).meterMin)
    {
        (**mH).meterMax = (**mH).meterMin;
        HiliteMeter(mH, 255); /* inactive - see IM 1-327 */
        return;
    }
    (**mH).meterMax = maxValue;
    if ((**mH).meterValue > maxValue)
        (**mH).meterValue = maxValue;
    SetDisplayDivisor(mH);
    DrawSet(mH);
    DrawNeedle(mH);
    ShowValue(mH, FALSE);
    SetPort(savePort);
} /* end SetMtrMax() */

long GetMtrMax(mH)
MeterHandle mH;
{
    if (!ValidMeter(mH)) return(0L);
    return(**mH).meterMax;
} /* end GetMtrMax() */

void SetMRefCon(mH, refCon)

```

```

MeterHandle mH;
long refCon;
{
    if (!ValidMeter(mH)) return;
    (**mH).meterRfCon = refCon;
} /* end SetMRefCon() */

long GetMRefCon(mH)
MeterHandle mH;
{
    if (!ValidMeter(mH)) return(0L);
    return(**mH).meterRfCon;
} /* end GetMRefCon() */

void SetNeedleValue(mH, value)
MeterHandle mH;
long value;
{
    GrafPtr savePort;

    if (!ValidMeter(mH)) return;
    if ((**mH).needleValue == value) return;
    GetPort(&savePort);
    SetPort(**mH).meterOwner);
    DrawNeedle(mH); /* erases old needle */
    /* pin value to range min:max */
    if (value < (**mH).meterMin)
        value = (**mH).meterMin;
    else if (value > (**mH).meterMax)
        value = (**mH).meterMax;
    (**mH).needleValue = value;
    DrawNeedle(mH);
    ShowValue(mH, TRUE);
    SetPort(savePort);
} /* end SetNeedleValue() */

long GetNeedleValue(mH)
MeterHandle mH;
{
    if (!ValidMeter(mH)) return(0L);
    return(**mH).needleValue;
} /* end GetNeedleValue() */

long GetDisplayDivisor(mH)
MeterHandle mH;
{
    if (!ValidMeter(mH)) return(0L);
    return(**mH).displayDivisor;
} /* end GetDisplayDivisor() */

Boolean ValidMeter(mH)
MeterHandle mH;
{
    if (mH && (**mH).hasMeterID == HASMETER)
        return(TRUE);
    return(FALSE);
} /* end ValidMeter() */

int GetMaxTitlePixels(mH)
MeterHandle mH;
{
    if (!ValidMeter(mH)) return(0);
    return(**mH).meterRect.right -
        (**mH).meterRect.left - 4;
} /* end GetMaxTitlePixels() */

/* A meter, like a scroll bar, is very attached to its owning
window. Only MeterSnapshot() will allow you to draw a meter in
another grafPort. Set the port, and set the font in the port
to something appropriate for the meter title before call.*/
void MeterSnapshot(mH)
MeterHandle mH;
{
    int i;

```

```

Byte mVis, mHilite;

if (!ValidMeter(mH)) return;
mVis = (**mH).meterVis;
mHilite = (**mH).meterHilite;
(**mH).meterVis = TRUE;
(**mH).meterHilite = 0;
DrawMeterBackground(mH);
for (i = inUpButton; i <= inPageDown; ++i)
    DrawTriangle(mH, i);
DrawNeedle(mH);
DrawSet(mH);
ShowValue(mH, TRUE);
(**mH).meterVis = mVis;
(**mH).meterHilite = mHilite;
} /* end MeterSnapshot() */

/* Functions used only in this file from here on */

/* Load up a table of sines to avoid going off on a tangent
(sorry). If this table were entered by hand, <math.h> would
not be needed. */
void InitMeters()
{
    int i;
    double x;

    for (i = 0; i <= 90; ++i)
    {
        x = ((double)i*PI)/180.0; /* convert degrees to radians */
        sineTable[i] = (int)(sin(x)*1000.0);
    }
} /* end InitMeters() */

/* The real fun - create the meter background picture and set
up points and rects needed for drawing it. We could also have
created the background in a drafting program and just scale it
to the meter's rectangle, but this wouldn't be as precise or
as easy to modify */
void CreateMeterBackground(mH)
MeterHandle mH;
{
    int i, j, k, h, v, posInc;
    long theSin, theCos;
    long radius, xo, yo, lowestTickH, lowestTickV;
    Rect boundRect, theClipRect,
        tempRect, innerCircle;

    GrafPtr savePort;
    RgnHandle clpRgn, dialRgn, innerTickRgn;
    PenState pState;
    PicHandle tempPicH;
    PolyHandle tempPoly;
    Point c, l, r; /* centre, left, right of polygons */

    GetPort(&savePort);
    SetPort((**mH).meterOwner);
    /* adjust meterRect - meter proportions are fixed,
meterRect.bottom is ignored */
    boundRect = (**mH).meterRect;
    if (boundRect.right - boundRect.left < MINWIDTH)
        boundRect.right = boundRect.left + MINWIDTH;
    boundRect.bottom = boundRect.top + ((boundRect.right -
        boundRect.left)*TVHRATIO)/1000;
    tempRect = boundRect;
    boundRect.bottom += TITLEHEIGHT;
    (**mH).meterRect = boundRect;
    /* first adjust the clip; */
    clpRgn = NewRgn();
    dialRgn = NewRgn();
    innerTickRgn = NewRgn();
    GetClip(clpRgn);
    /* draw meter in rectangle starting at 0,0 for convenience */
    OffsetRect(&boundRect, -boundRect.left, -boundRect.top);
    OffsetRect(&tempRect, -tempRect.left, -tempRect.top);

```

```

theClipRect = boundRect;
theClipRect.right += 100;
theClipRect.bottom += 200;
ClipRect(&theClipRect);
GetPenState(&pState);

tempPicH = OpenPicture(&boundRect);
PenNormal();
/* draw black rounded rect */
FillRoundRect(&tempRect, tempRect.right/10,
    tempRect.right/10, black);
i = tempRect.bottom;
/* and frame two white circles */
tempRect.bottom = tempRect.right; /* square it off */
InsetRect(&tempRect, tempRect.right/15,
    tempRect.right/15);
j = tempRect.bottom;
FillOval(&tempRect, white);
InsetRect(&tempRect, 3, 3);
radius = (tempRect.right - tempRect.left)/2;
(**mH).dialCentre.h = tempRect.left + radius;
(**mH).dialCentre.v = tempRect.top + radius;
FrameOval(&tempRect);
OpenRgn();
FrameOval(&tempRect);
CloseRgn(dialRgn);
k = radius - (radius*INNERPCT + 50L)/100L;
InsetRect(&tempRect, k, k);
innerCircle = tempRect;
OpenRgn();
FrameOval(&tempRect);
CloseRgn(innerTickRgn);
DiffRgn(dialRgn, innerTickRgn, dialRgn); /* That
made a ring */
DisposeRgn(innerTickRgn);
/* white out rect for control triangles */
tempRect.top = j + j/20; /* j = bottom of
larger dial circle */
tempRect.left = (**mH).dialCentre.h - radius;
tempRect.bottom = i - j/20; /* i = bottom of rounded rect */
tempRect.right = (**mH).dialCentre.h + radius;
i = j = tempRect.right - tempRect.left;
i = (i/4)*4;
tempRect.left += (j - i)/2;
tempRect.right = tempRect.left + i;
FillRect(&tempRect, white);
/* Put lines between the control triangles. */
MoveTo(tempRect.left + i/4, tempRect.top);
Line(0, tempRect.bottom - tempRect.top);
MoveTo(tempRect.left + i/2, tempRect.top);
Line(0, tempRect.bottom - tempRect.top);
MoveTo(tempRect.left + i/4 + i/2, tempRect.top);
Line(0, tempRect.bottom - tempRect.top);
tempRect.right = tempRect.left + i/4;
(**mH).pageDownRect = tempRect;
OffsetRect(&tempRect, i/4, 0);
(**mH).incDownRect = tempRect;
OffsetRect(&tempRect, i/4, 0);
(**mH).incUpRect = tempRect;
OffsetRect(&tempRect, i/4, 0);
(**mH).pageUpRect = tempRect;
SetClip(dialRgn);
/* dialRgn is the ring for the ticks */
PenSize(2,2);
/* Draw the major ticks around circumference of dial */
h = (**mH).dialCentre.h;
v = (**mH).dialCentre.v;
if (NUMTICKS >= 3)
{
    posInc = 2500/(NUMTICKS - 1);
    for (j = -1250; j <= 1250;
        j += posInc)
    {
        i = j/10;

```

```

    GetSinAndCos(i, &theSin, &theCos);
    xo = (radius*theSin + 500L)/1000L + h;
    yo = (-radius*theCos + 500L)/1000L + v;
    MoveTo(h - 1, v - 1);
    LineTo(xo - 1, yo - 1);
}
)
/* and now the minor ticks */
PenSize(1,1);
if (NUMMINORTICKS >= 3)
{
    posInc = 250/(NUMMINORTICKS - 1);
    for (j = -1250; j <= 1250;
        j += posInc)
    {
        i = j/10;
        /* determine sine and cosine to use */
        GetSinAndCos(i, &theSin, &theCos);
        xo = (radius*theSin + 500L)/1000L + h;
        yo = (-radius*theCos + 500L)/1000L + v;
        MoveTo(h - 1, v - 1);
        LineTo(xo - 1, yo - 1);
    }
}

ClipRect(&theClipRect);
DisposeRgn(dialRgn);
/* White out centre - needed for clipboard copying.*/
FillOval(&innerCircle, white);
/* Put small black circle in centre of dial.*/
tempRect.top = (**mH).dialCentre.v - CENTRERADIUS;
tempRect.left = (**mH).dialCentre.h - CENTRERADIUS;
tempRect.bottom = (**mH).dialCentre.v + CENTRERADIUS;
tempRect.right = (**mH).dialCentre.h + CENTRERADIUS;
FillOval(&tempRect, black);
/* Set up and draw displayRect just below dial centre. */
lowestTickH = ((xo - h)*INNERPCT)/100L + h;
lowestTickV = ((yo - v)*INNERPCT)/100L + v;
tempRect.right = lowestTickH - 2160/((**mH).meterRect.right
    - (**mH).meterRect.left);
tempRect.left = tempRect.right - 2*(tempRect.right - h);
tempRect.top = (int)lowestTickV;
tempRect.bottom = tempRect.top + 16;
(**mH).displayRect = tempRect;
AdjustDisplay(mH);
tempRect = (**mH).displayRect;
OffsetRect(&tempRect, -1, 0);
FrameRect(&tempRect);
MoveTo(tempRect.left + 1, tempRect.bottom);
LineTo(tempRect.right, tempRect.bottom);
MoveTo(tempRect.right, tempRect.top + 1);
LineTo(tempRect.right, tempRect.bottom);
InsetRect(&tempRect, 1, 1);
i = (**mH).meterRect.left;
j = (**mH).meterRect.top;
OffsetRect(&tempRect, i, j);
(**mH).displayRect = tempRect;
SetDisplayDivisor(mH);
ClosePicture();

(**mH).backgroundH = tempPicH;
/* Define up and down arrows as triangular polygons.*/
/* page up poly */
tempRect = (**mH).pageUpRect;
l.h = tempRect.left + (tempRect.right - tempRect.left)
    - tempRect.left)/8;
r.h = tempRect.right - (tempRect.right - tempRect.left)/8;
i = r.h - l.h; /* side length of triangle */
i = (i/2)*2;
r.h = l.h + i;
j = (i*PGPOLYHEIGHT + 1000)/2000; /* vertical
    height, 1/2side * root3 */
c.v = tempRect.top + (tempRect.bottom - tempRect.top - j)/2;

```

```

c.h = tempRect.left + (tempRect.right - tempRect.left)/2;
l.v = c.v + j;
r.v = c.v + j;
tempPoly = OpenPoly();
    MoveTo(c.h, c.v);
    LineTo(l.h, l.v);
    LineTo(r.h, r.v);
    LineTo(c.h, c.v);
ClosePoly();
(**mH).pageUpH = tempPoly;
/* inc up poly */
tempRect = (**mH).incUpRect;
l.h = tempRect.left + (tempRect.right - tempRect.left)/4;
r.h = tempRect.right - (tempRect.right - tempRect.left)/4;
i = r.h - l.h; /* side length of triangle */
i = (i/2)*2;
r.h = l.h + i;
j = (i*INCPOLYHEIGHT + 1000)/2000; /* vertical
    height, 1/2side * root3 */
c.v = tempRect.top + (tempRect.bottom - tempRect.top - j)/2;
c.h = tempRect.left + (tempRect.right - tempRect.left)/2;
l.v = c.v + j;
r.v = c.v + j;
tempPoly = OpenPoly();
    MoveTo(c.h, c.v);
    LineTo(l.h, l.v);
    LineTo(r.h, r.v);
    LineTo(c.h, c.v);
ClosePoly();
(**mH).incUpH = tempPoly;
/* inc down poly */
tempRect = (**mH).incDownRect;
l.h = tempRect.left + (tempRect.right - tempRect.left)/4;
r.h = tempRect.right - (tempRect.right - tempRect.left)/4;
i = r.h - l.h; /* side length of triangle */
i = (i/2)*2;
r.h = l.h + i;
j = (i*INCPOLYHEIGHT + 1000)/2000; /* vertical
    height, 1/2side * root3 */
c.v = tempRect.top + (tempRect.bottom - tempRect.top - j)/2 + j;
c.h = tempRect.left + (tempRect.right - tempRect.left)/2;
l.v = c.v - j;
r.v = c.v - j;
tempPoly = OpenPoly();
    MoveTo(c.h, c.v);
    LineTo(l.h, l.v);
    LineTo(r.h, r.v);
    LineTo(c.h, c.v);
ClosePoly();
(**mH).incDownH = tempPoly;
/* page down poly */
tempRect = (**mH).pageDownRect;
l.h = tempRect.left + (tempRect.right - tempRect.left)/8;
r.h = tempRect.right - (tempRect.right - tempRect.left)/8;
i = r.h - l.h; /* side length of triangle */
i = (i/2)*2;
r.h = l.h + i;
j = (i*PGPOLYHEIGHT + 1000)/2000; /* vertical
    height, 1/2side * root3 */
c.v = tempRect.top + (tempRect.bottom - tempRect.top - j)/2 + j;
c.h = tempRect.left + (tempRect.right - tempRect.left)/2;
l.v = c.v - j;
r.v = c.v - j;
tempPoly = OpenPoly();
    MoveTo(c.h, c.v);
    LineTo(l.h, l.v);
    LineTo(r.h, r.v);
    LineTo(c.h, c.v);
ClosePoly();
(**mH).pageDownH = tempPoly;

```

```

/* Set up needle and set point parameters. */
radius = (radius*(INNERPCT-2L) + 50L)/100;
(**mH).needleLong = radius;
(**mH).setLong = radius;
(**mH).needleShort = (lowestTickV - (**mH).dialCentre.v)/2;
(**mH).setShort = (**mH).needleShort;
(**mH).halfNeedleWidth = radius/HALFNEEDLERATIO;

/* Move rects and points to proper position. */
i = (**mH).meterRect.left;
j = (**mH).meterRect.top;
OffsetRect(&(**mH).incDownRect, i,j);
OffsetRect(&(**mH).incUpRect, i,j);
OffsetRect(&(**mH).pageDownRect, i,j);
OffsetRect(&(**mH).pageUpRect, i,j);
(**mH).dialCentre.h += i;
(**mH).dialCentre.v += j;
/* clean up on the way out */
SetPenState(&pState);
SetClip(c1pRgn);
DisposeRgn(c1pRgn);
SetPort(savePort);
} /* end CreateMeterBackground() */

void DrawMeterBackground(mH)
MeterHandle mH;
{
    PicHandle tempPH = (**mH).backgroundH;
    Rect tempRect;
    PenState pState;

    GetPenState(&pState);
    PenNormal();
    tempRect = (**mH).meterRect;
    EraseRect(&tempRect);
    if (**mH).meterVis)
    {
        if (**mH).meterHilite != 255)
            DrawPicture(tempPH, &tempRect);
        else
            DrawInactiveMeter(mH);
        DrawMeterTitle(mH);
    }
    SetPenState(&pState);
} /* end DrawMeterBackground() */

/* Title is centered and truncated if necessary, in whatever
font is current. Drawing is clipped to the intersection of the
title rect and the current clip region. */
void DrawMeterTitle(mH)
MeterHandle mH;
{
    int i,j, height;
    Rect tempRect;
    char title[32];
    FontInfo tempInfo;
    PenState pState;
    RgnHandle oldClipRgn, titleClipRgn;
    ControlHandle cHH, cHV;
    char *mTPtr = (**mH).title;

    GetFontInfo(&tempInfo);
    height = tempInfo.ascent + tempInfo.descent
        + tempInfo.leading;
    /* Copy title - text can't be referenced by
    an unlocked handle when calling DrawString(). */
    for (i = 0; i < 31; ++i)
        title[i] = mTPtr[i];
    /* Draw shadowed title box, effective height =
    TITLEHEIGHT-4, just below black rounded rect*/
    tempRect.left = (**mH).meterRect.left + 2;
    tempRect.right = (**mH).meterRect.right - 2;
    tempRect.bottom = (**mH).meterRect.bottom - 2;
    tempRect.top = tempRect.bottom - TITLEHEIGHT + 4;

```

```

FrameRect(&tempRect);
MoveTo(tempRect.left + 1, tempRect.bottom);
LineTo(tempRect.right, tempRect.bottom);
MoveTo(tempRect.right, tempRect.top + 1);
LineTo(tempRect.right, tempRect.bottom);
i = StringWidth(title);
j = tempRect.right - tempRect.left;
if ((j - i) < 2) /* title too long - truncate */
{
    while (StringWidth(title) > j - 2)
        title[0] -= 1;
    i = StringWidth(title);
}
GetFontInfo(&tempInfo);
height = tempInfo.ascent + tempInfo.descent
    + tempInfo.leading;
MoveTo(tempRect.left + (j - i)/2, tempRect.top +
    (tempRect.bottom - tempRect.top)/2 - height/2
    + tempInfo.ascent);
oldClipRgn = NewRgn();
titleClipRgn = NewRgn();
GetClip(oldClipRgn);
RectRgn(titleClipRgn, &tempRect);
SectRgn(titleClipRgn, oldClipRgn, titleClipRgn);
SetClip(titleClipRgn);
GetPenState(&pState);
PenNormal();

DrawString(title);

SetPenState(&pState);
SetClip(oldClipRgn);
DisposeRgn(oldClipRgn);
DisposeRgn(titleClipRgn);
} /* end DrawMeterTitle() */

/* The four triangles (two small, two large)
correspond to the inc and page parts of a scroll bar. */

void DrawTriangle(mH, partCode)
MeterHandle mH;
int partCode;
{
    PenState pState;
    PolyHandle tempPoly;

    if (**mH).meterHilite == 255) return;
    if (partCode == inUpButton)
        tempPoly = (**mH).incUpH;
    else if (partCode == inDownButton)
        tempPoly = (**mH).incDownH;
    else if (partCode == inPageUp)
        tempPoly = (**mH).pageUpH;
    else if (partCode == inPageDown)
        tempPoly = (**mH).pageDownH;
    else
        return;
    GetPenState(&pState);
    PenNormal();
    OffsetPoly(tempPoly, (**mH).meterRect.left,
        (**mH).meterRect.top);
    if (partCode == (**mH).meterHilite)
    {
        OffsetPoly(tempPoly, -1, -1);
        FillPoly(tempPoly, black);
    }
    else
    {
        /* neat way to shadow ANY polygon */
        FillPoly(tempPoly, black);
        FramePoly(tempPoly);
        OffsetPoly(tempPoly, -1, -1);
        ErasePoly(tempPoly);
        FramePoly(tempPoly);
    }
}

```

```

OffsetPoly(tempPoly, -(**mH).meterRect.left + 1,
           -(**mH).meterRect.top + 1);
SetPenState(&pState);
} /* end DrawTriangle() */

void DrawNeedle(mH)
MeterHandle mH;
{
    long    l1, l2, l3, xa, xb, xc, xd;
    long    theSin, theCos, ya, yb, yc, yd;
    int     position;
    int     h = (**mH).dialCentre.h;
    int     v = (**mH).dialCentre.v;
    PenState pState;
    PolyHandle needlePoly;

    if (!(**mH).meterVis) ||
        (**mH).meterHilite == 255) return;
    GetPenState(&pState);
    PenMode(patXor);
    PenPat(black);
    PenSize(1,1);

    /* Determine position of needle based on value. Position
    is the angle in degrees measured from 0 = straight up,
    increasing clockwise, range -125:+125 */
    position = (((**mH).needleValue
                - (**mH).meterMin)*250L)
              /((**mH).meterMax - (**mH).meterMin) - 125L;
    if (position < -125)
        position = -125;
    if (position > 125)
        position = 125;
    /* Determine sine and cosine to use. */
    GetSinAndCos(position, &theSin, &theCos);
    /* Calculate four vertices of needle indicator by
    changing from polar to rectangular coord's and then trans-
    lating over to dial centre. */
    l1 = (**mH).needleLong;
    l2 = (**mH).needleShort;
    l3 = (**mH).halfNeedleWidth;
    xa = (-12*theSin + 500L)/1000L + h;
    ya = (12*theCos + 500L)/1000L + v;
    xb = (11*theSin + 500L)/1000L + h;
    yb = (-11*theCos + 500L)/1000L + v;
    xc = (-13*theCos + 500L)/1000L + h;
    yc = (-13*theSin + 500L)/1000L + v;
    xd = -xc + 2*h;
    yd = -yc + 2*v;

    needlePoly = OpenPoly();
    MoveTo(xa, ya);
    LineTo(xc, yc);
    LineTo(xb, yb);
    LineTo(xd, yd);
    LineTo(xa, ya);
    ClosePoly();
    PaintPoly(needlePoly);
    KillPoly(needlePoly);

    SetPenState(&pState);
} /* end DrawNeedle() */

```

```

void DrawSet(mH)
MeterHandle mH;
{
    long    l1, l2, l3, xa, xb;
    long    theSin, theCos, ya, yb;
    int     position;
    int     thePenSize = (**mH).halfSetWidth*2;
    int     h = (**mH).dialCentre.h;
    int     v = (**mH).dialCentre.v;
    PenState pState;

```

```

    if (!(**mH).meterVis) ||
        (**mH).meterHilite == 255) return;
    GetPenState(&pState);
    PenMode(patXor);
    PenPat(black);
    PenSize(thePenSize, thePenSize);

    /* Determine position of set indicator from value. */
    position = (((**mH).meterValue - (**mH).meterMin)*250L)
              /((**mH).meterMax - (**mH).meterMin) - 125L;
    if (position < -125)
        position = -125;
    if (position > 125)
        position = 125;

```

```

    /* Determine sine and cosine to use */
    GetSinAndCos(position, &theSin, &theCos);

    /* Calculate two vertices of position indicator */
    l1 = (**mH).setLong;
    l2 = (**mH).setShort;
    l3 = (**mH).halfSetWidth;
    xa = (-12*theSin + 500L)/1000L - HALFSETWIDTH + h;
    ya = (12*theCos + 500L)/1000L - HALFSETWIDTH + v;
    xb = (11*theSin + 500L)/1000L - HALFSETWIDTH + h;
    yb = (-11*theCos + 500L)/1000L - HALFSETWIDTH + v;

```

```

    MoveTo(xa, ya);
    LineTo(xb, yb);

    SetPenState(&pState);
} /* end DrawSet() */

```

```

void ShowValue(mH, needle)
MeterHandle mH;
Boolean needle;
{
    int     i;
    int     oldFont, oldSize;
    Style    oldStyle;
    PenState pState;
    long    theValue;
    Rect     tempRect;
    char     pNumStr[16]; /* pascal */

    if (!(**mH).meterVis) ||
        (**mH).meterHilite == 255) return;
    GetPenState(&pState);
    PenNormal();
    oldFont = thePort->txFont;
    oldSize = thePort->txSize;
    oldStyle = thePort->txFace;
    TextFont(geneva);
    TextSize(12);
    TextFace(' ');
    if (needle)
        theValue = (**mH).needleValue;
    else
        theValue = (**mH).meterValue;
    /* Truncate value to fit display box. */
    if ((**mH).displayDivisor > 1L)
        theValue /= (**mH).displayDivisor;
    tempRect = (**mH).displayRect;
    NumToString(theValue, pNumStr);
    i = StringWidth(pNumStr);
    EraseRect(&tempRect);
    MoveTo(tempRect.right - i - 4, tempRect.top +
            (tempRect.bottom - tempRect.top)/2 + TEXTV0FFSET);
    DrawString(pNumStr);
    TextFont(oldFont);
    TextSize(oldSize);
    TextFace(oldStyle);
    SetPenState(&pState);
} /* end ShowValue() */

```

```

void DrawInactiveMeter(mH)
MeterHandle mH;
{
    Rect tempRect;

    tempRect = (**mH).meterRect;
    tempRect.bottom -= TITLEHEIGHT;
    FrameRoundRect(&tempRect, tempRect.right/10,
        tempRect.bottom/10);
} /* end DrawInactiveMeter() */

/* Ensure that digital display shows at least MINDIGITS and at
most MAXDIGITS plus sign - called only during creation of
background meter picture. */
void AdjustDisplay(mH)
MeterHandle mH;
{
    int oldFont, oldSize;
    Style oldStyle;
    int displayWidth =
        (**mH).displayRect.right
        - (**mH).displayRect.left;

    int minusW, num4W;
    int numDisplayDigits, wantedWidth, halfWidthDiff;

    oldFont = thePort->txFont;
    oldSize = thePort->txSize;
    oldStyle = thePort->txFace;
    TextFont(geneva);
    TextSize(12);
    TextFace(' ');
    minusW = CharWidth('-');
    num4W = CharWidth('4');
    numDisplayDigits = (displayWidth - minusW - 8)/num4W;
    if (numDisplayDigits < MINDIGITS) /* too small */
    {
        wantedWidth = 8 + minusW + num4W*MINDIGITS;
        halfWidthDiff = (wantedWidth - displayWidth + 1)/2;
        (**mH).displayRect.left -= halfWidthDiff;
        (**mH).displayRect.right += halfWidthDiff;
    }
    else if (numDisplayDigits > MAXDIGITS) /* too big */
    {
        wantedWidth = 8 + minusW + num4W*MAXDIGITS;
        halfWidthDiff = (displayWidth - wantedWidth - 1)/2;
        (**mH).displayRect.left += halfWidthDiff;
        (**mH).displayRect.right -= halfWidthDiff;
    }
    TextFont(oldFont);
    TextSize(oldSize);
    TextFace(oldStyle);
} /* end AdjustDisplay() */

/* Set displayDivisor to appropriate power of ten so that as
many digits as possible are shown in the display. Called when
creating meter and also when setting meter max or min */
void SetDisplayDivisor(mH)
MeterHandle mH;
{
    int oldFont, oldSize;
    Style oldStyle;
    int displayWidth = (**mH).displayRect.right
        - (**mH).displayRect.left + 2;
    int minusW, num4W, numDisplayDigits,
        wantToDisplay, power;
    long absMin, absMax;
    char pNumStr[16]; /* pascal */

    oldFont = thePort->txFont;
    oldSize = thePort->txSize;
    oldStyle = thePort->txFace;
    TextFont(geneva);
    TextSize(12);

```

```

    TextFace(' ');
    minusW = CharWidth('-');
    num4W = CharWidth('4');
    numDisplayDigits = (displayWidth - minusW - 8)/num4W;
    absMin = ((**mH).meterMin < 0) ?
        - (**mH).meterMin : (**mH).meterMin;
    absMax = ((**mH).meterMax < 0) ?
        - (**mH).meterMax : (**mH).meterMax;
    if (absMin > absMax)
        absMax = absMin;
    NumToString(absMax, pNumStr);
    wantToDisplay = (int)(pNumStr[0]);
    if (wantToDisplay > numDisplayDigits) /* too many,
        must scale value down */
    {
        /* divisor = 10 raised to power
        (wantToDisplay - numDisplayDigits) */
        power = wantToDisplay - numDisplayDigits;
        (**mH).displayDivisor = 10L;
        while (-power > 0)
            (**mH).displayDivisor *= 10L;
    }
    else /* no scaling necessary */
        (**mH).displayDivisor = 1L;
    TextFont(oldFont);
    TextSize(oldSize);
    TextFace(oldStyle);
} /* end SetDisplayDivisor() */

```

/* Position is interpreted as an angle in degrees. The values are retrieved from sineTable[], which is an array of integers recording sines from 0 to 90 degrees, scaled up by 1000. Note zero degrees is straight up, & angles increase clockwise. */

```

void GetSinAndCos(position, theSin, theCos)
int position;
long *theSin, *theCos;
{
    if (position < -125 || position > 125)
    {
        *theSin = 0L;
        *theCos = 0L;
        return;
    }
    else if (-125 <= position && position <= -90)
    {
        *theSin = -sineTable[position + 180];
        *theCos = -sineTable[-position - 90];
    }
    else if (-90 < position && position <= 0)
    {
        *theSin = -sineTable[-position];
        *theCos = sineTable[position + 90];
    }
    else if (0 < position && position <= 90)
    {
        *theSin = sineTable[position];
        *theCos = sineTable[90 - position];
    }
    else /* 90:125 */
    {
        *theSin = sineTable[180 - position];
        *theCos = -sineTable[position - 90];
    }
} /* end GetSinAndCos() */

```

Listing: MeterMain.C

/* MeterMain.c - test rig for designing meters, illustrating the interaction between documents, windows, and controls by Ken Earle, 1989

ThinkC 3.0 settings:

Options...; <MacHeaders>, Check Pointers, Require Prototypes

Set Project Type...; File Type APPL, Creator whatever


```

Files: MacTraps
      Math
      Meter.c      MeterMain.c
Headers: <MacHeaders> <math.h> "Meter.h"
Resources: none (What? Blasphemy!)
*/

#include "Meter.h"
#define APPLEID      15
#define FILEID       16
#define EDITID       17
#define QUIT         12
#define HSCROLL      1L /* scroll bar ID */
#define VSCROLL      2L
#define SCROLLSIZE   15
#define DOCHEIGHT     720 /* 10" x 72 */
#define DOCWIDTH      576 /* 8" x 72 */
#define MAXWINDOWWIDTH (DOCWIDTH + SCROLLSIZE)
#define MAXWINDOWHEIGHT (DOCHEIGHT + SCROLLSIZE)
#define DISPLAYBOXWIDTH 72 /* <= 0 removes the display
                           and disables DrawDisplayBox() */
/* display modes are used by DrawDisplayBox() */
enum DisplayMode
{
    forWindow, forMeter, forThumb, forDebug = 300
    /* forces to 16-bit size */
};
typedef enum DisplayMode DisplayMode;

/* useful globals */
WindowRecord wRecord1, wRecord2;
RgnHandle    updateRgn;
MenuHandle appleMenu, fileMenu, editMenu;
/* handy globals, bad style */
Rect dragRect = { 0, 0, 1024, 1024 };
char wTitle1[] = "\P Rear Window";
char wTitle2[] = "\P Front Window";
char mTitle1[] = "\P First Meter";
char mTitle2[] = "\P Second Meter";
char mTitle3[] = "\P Third Meter";
char mTitle4[] = "\P Fourth Meter";

/* Prototypes for functions defined in this file */
int main(void);
void DoStandardInits(void);
void MakeMenus(void);
Boolean DoMenu(long);
void MakeTwoWindows(void);
void MakeTwoMeters(WindowPtr, int);
void DitherMeters(void);
void DoContent(WindowPtr, EventRecord*);
void UpdateContents(WindowPtr, Boolean);
void DrawDisplayBox(WindowPtr, void**,
                    DisplayMode, int, int);
void DoZoomWindow (WindowPtr, int);
void DoGrowWindow(EventRecord*);
void DoCloseWindow(void);
Boolean AMeterWindow(WindowPtr);
void GetScrollHandles(WindowPtr, ControlHandle*, Control-
Handle*);
void LocalToDoc(Point*);
void MeterProc(MeterHandle, int);
pascal void ScrollProc(ControlHandle, int);
void ThumbControl(ControlHandle, Point);
void PutMetersOnClip(WindowPtr);

main()
{
    int part;
    EventRecord myEvent;
    WindowPtr whichWindow;
    GrafPtr savePort;
    MeterHandle mH;
    ControlHandle cH;

```

```

    Rect cRect; /* see activateEvt */
    RgnHandle tempRgn; /* see updateEvt */
    Boolean quitting = FALSE;

    DoStandardInits();
    updateRgn = NewRgn();
    MakeMenus();
    MakeTwoWindows();

    do /* while not quitting */
    {
        do /* while no next event */
        {
            SystemTask();
            DitherMeters();
        } while (!GetNextEvent(everyEvent, &myEvent));
        switch (myEvent.what)
        {
            case mouseDown:
                switch (part = FindWindow(myEvent.where, &whichWindow))
                {
                    case inDesk:
                        SysBeep(2);
                        break;
                    case inMenuBar:
                        quitting =
                            DoMenu(MenuSelect(myEvent.where));
                    case inSysWindow:
                        SystemClick(&myEvent, whichWindow);
                        break;
                    case inDrag:
                        if (AMeterWindow(whichWindow))
                            DragWindow(whichWindow, myEvent.where, &dragRect);
                        break;
                    case inZoomIn:
                    case inZoomOut:
                        if (AMeterWindow(whichWindow))
                            if (TrackBox(whichWindow, myEvent.where, part))
                                DoZoomWindow(whichWindow, part);
                        break;
                    case inGrow:
                        if (AMeterWindow(whichWindow))
                            DoGrowWindow(&myEvent);
                        break;
                    case inGoAway:
                        if (AMeterWindow(whichWindow))
                            if (TrackGoAway(whichWindow, myEvent.where))
                                DoCloseWindow();
                        break;
                    case inContent:
                        if (whichWindow != FrontWindow())
                            SelectWindow(whichWindow);
                        else
                            if (AMeterWindow(whichWindow))
                                DoContent(whichWindow, &myEvent);
                        break;
                    default:
                        break;
                } /* switch FindWindow */
            break;
            case keyDown:
            case autoKey:
            break;
            case activateEvt:
                whichWindow = (WindowPtr)myEvent.message;
                if (AMeterWindow(whichWindow))
                {
                    mH = (MeterHandle)((WindowPeek)whichWindow->refCon);
                    cH = ((WindowPeek)whichWindow->controlList);
                    if (myEvent.modifiers & activeFlag)
                    {
                        SetPort(whichWindow);
                        ClipRect(&(whichWindow->portRect));

```

```

InvalRect(&(whichWindow->portRect));
DrawGrowIcon(whichWindow);
if (cH)
{
    while (cH)
    {
        ShowControl(cH);
        cRect = (**cH).controlRect;
        ValidRect(&cRect);
        cH = (**cH).nextControl;
    }
}
if (ValidMeter(mH))
{
    while (mH)
    {
        /* activate meters */
        (**mH).meterHilite = 0;
        mH = (**mH).nextMeter;
    }
}
cRect = thePort->portRect;
cRect.bottom -= SCROLLSIZE;
cRect.right -= SCROLLSIZE;
ClipRect(&cRect);
} /* if activate */
else
{
    GetPort(&savePort);
    SetPort(whichWindow);
    ClipRect(&(whichWindow->portRect));
    InvalRect(&(whichWindow->portRect));
    DrawGrowIcon(whichWindow);
    if (cH)
    {
        while(cH)
        {
            HideControl(cH);
            cH = (**cH).nextControl;
        }
    }
    if (ValidMeter(mH))
    {
        while (mH)
        {
            /* deactivate meters */
            (**mH).meterHilite = 255;
            mH = (**mH).nextMeter;
        }
    }
    SetPort(savePort);
} /* else deactivate */
} /* if our window */
break;
case updateEvt:
    whichWindow = (WindowPtr)myEvent.message;
    if (AMeterWindow(whichWindow))
    {
        GetPort(&savePort);
        SetPort( whichWindow );
        ClipRect(&whichWindow->portRect);
        BeginUpdate( whichWindow );
        EraseRect(&whichWindow->portRect);
        DrawGrowIcon(whichWindow);
        DrawControls(whichWindow);
        DrawDisplayBox(whichWindow, 0L, forWindow, 0, 0);
        /* Take the scrolls out of updateRgn, because
        UpdateContents() sets the clip region to the
        (global) updateRgn. */
        CopyRgn(whichWindow->visRgn, updateRgn);
        tempRgn = NewRgn();
        SetRectRgn(tempRgn,
            whichWindow->portRect.left,
            whichWindow->portRect.bottom-SCROLLSIZE,

```

```

            whichWindow->portRect.right,
            whichWindow->portRect.bottom);
        DiffRgn(updateRgn, tempRgn, updateRgn);
        SetRectRgn(tempRgn,
            whichWindow->portRect.right-SCROLLSIZE,
            whichWindow->portRect.top,
            whichWindow->portRect.right,
            whichWindow->portRect.bottom-SCROLLSIZE);
        DiffRgn(updateRgn, tempRgn, updateRgn);
        DisposeRgn(tempRgn);
        /* Redraw the "true contents" of the window. */
        UpdateContents(whichWindow,
            (whichWindow == FrontWindow()));
        EndUpdate( whichWindow );
        SetPort(savePort);
    }
    break;
default:
    break;
} /* switch myEvent.what */
} while (!quitting);
if (FrontWindow())
    PutMetersOnClip(FrontWindow());
} /* end main() */

```

```

void DoStandardInits()
{
    MoreMasters();
    MoreMasters();
    MoreMasters();
    InitGraf(&thePort);
    InitFonts();
    FlushEvents(everyEvent, 0);
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs(0L);
    InitCursor();
} /* end DoStandardInits() */

```

```

void MakeMenus()
{
    appleMenu = NewMenu(APPLEID, "\P\24");
    AppendMenu(appleMenu, "\P(About nothing...;(-)");
    AddResMenu(appleMenu, 'DRVr');
    InsertMenu(appleMenu, 0);

    fileMenu = NewMenu(FILEID, "\PFile");
    AppendMenu(fileMenu, "\PNew;Open...;\
(-;Close;Save;Save as...;Revert;(-;\
Page Setup...;Print...;(-;Quit/Q");
    InsertMenu(fileMenu, 0);

    editMenu = NewMenu(EDITID, "\PEdit");
    AppendMenu(editMenu, "\PUndo/Z;\
(-;Cut/X;Copy/C;Paste/V;Clear;(-;\
Show Clipboard;Select All");
    InsertMenu(editMenu, 0);

    DrawMenuBar();
} /* end MakeMenus() */

```

```

/* DoMenu returns TRUE when quit selected */
Boolean DoMenu(menuAndItem)
long menuAndItem;
{
    int theItem;
    char name[64];

    theItem = LoWord(menuAndItem);
    switch (HiWord(menuAndItem))
    {
        case APPLEID:
            if (theItem > 1)

```

```

    {
        if (FrontWindow())
            PutMetersOnClip(FrontWindow());
        GetItem(appleMenu, theItem, name);
        OpenDeskAcc(name);
    }
    break;
case FILEID:
    if (theItem == QUIT)
        return(TRUE);
    break;
case EDITID:
    break;
default:
    break;
}
HiliteMenu(0);
return(FALSE);
} /* end DoMenu() */

/* Make two windows right off the bat, each with a couple of
scroll bars and a couple of meters. The windows are sized to
fit the screen, but limited
to a maximum size of 8" by 10" to show the slight complica-
tions that arise when dealing with a fixed document size. The
trick thereafter is to ensure that you never stray outside the
document - see DoGrowWindow() etc */
void MakeTwoWindows()
{
    int i, topGap, winWidth, winHeight;
    int horRange, vertRange;
    Rect boundsRect, cRect, clpRect;
    WindowPtr wPtr;
    ControlHandle ch;
    WStateData *wsd;

    /* Old small screen, or something bigger?
topGap = menu bar height + 20 for the window frame */
    if ((screenBits.bounds.bottom
        - screenBits.bounds.top) < 343)
        topGap = MBarHeight + 20;
    else
        topGap = 60;
    /* topGap + window height + all offsets(1x10 here) + 4(at
the bottom) = screen height */
    winHeight = ((screenBits.bounds.bottom -
        screenBits.bounds.top - topGap - 14)
        <= MAXWINDOWHEIGHT) ?
        (screenBits.bounds.bottom -
        screenBits.bounds.top - topGap - 14) :
        MAXWINDOWHEIGHT;
    winWidth = ((screenBits.bounds.right
        - screenBits.bounds.left - 18)
        <= MAXWINDOWWIDTH) ?
        (screenBits.bounds.right -
        screenBits.bounds.left - 18) :
        MAXWINDOWWIDTH;
    SetRect(&boundsRect, 4, topGap, 4 + winWidth, topGap +
winHeight);
    /* Scroll bar ranges: range = amount of document
left unseen (honest!) */
    vertRange = MAXWINDOWHEIGHT
        - (boundsRect.bottom - boundsRect.top);
    vertRange = (vertRange > 0) ? vertRange : 0;
    horRange = MAXWINDOWWIDTH
        - (boundsRect.right - boundsRect.left);
    horRange = (horRange > 0) ? horRange : 0;
    /* Create two windows, offset slightly */
    for (i = 1; i <= 2; ++i)
    {
        if (i == 1)
            wPtr = NewWindow(&wRecord1, &boundsRect,
                wTitle1, TRUE, 8, -1L, TRUE, 0L);
        else

```

```

    {
        OffsetRect(&boundsRect, 10, 10);
        wPtr = NewWindow(&wRecord2, &boundsRect,
            wTitle2, TRUE, 8, -1L, TRUE, 0L);
    }
    SetPort(wPtr);
    ClipRect(&(thePort->portRect));
    /* Adjust zoom rectangles since document size is fixed. */
    wsd = (WStateData*) *(((WindowPeek)wPtr)->dataHandle);
    wsd->stdState = boundsRect;
    wsd->userState = boundsRect;
    /* horizontal bar */
    /* Note room is reserved at right of the bar for the
display box - see DrawDisplayBox(). */
    cRect.top = thePort->portRect.bottom - SCROLLSIZE;
    cRect.left = thePort->portRect.left - 1;
    cRect.bottom = thePort->portRect.bottom + 1;
    cRect.right = thePort->portRect.right
        - DISPLAYBOXWIDTH - 14;
    ch = NewControl(thePort, &cRect, "\P", TRUE,
        0, 0, horRange, scrollBarProc, HSCROLL);
    if (horRange == 0)
        HiliteControl(ch, 255);
    /* vertical bar */
    cRect.top = thePort->portRect.top - 1;
    cRect.left = thePort->portRect.right - SCROLLSIZE;
    cRect.bottom = thePort->portRect.bottom - 14;
    cRect.right = thePort->portRect.right + 1;
    ch = NewControl(thePort, &cRect, "\P", TRUE,
        0, 0, vertRange, scrollBarProc, VSCROLL);
    if (vertRange == 0)
        HiliteControl(ch, 255);
    clpRect = thePort->portRect;
    clpRect.bottom -= SCROLLSIZE;
    clpRect.right -= SCROLLSIZE;
    ClipRect(&clpRect);
    MakeTwoMeters(wPtr, i);
} /* end MakeTwoWindows() */

/* Create four meters, two in each window. Note that it is not
necessary to carry around all the meterhandles as globals
since they are attached to the window's refCon field as a
linked list by NewMeter(). */
void MakeTwoMeters(WindowPtr which)
{
    WindowPtr wPtr;
    int which;
    {
        Rect boundsRect;
        MeterHandle mH;

        if (which == 1)
        {
            boundsRect.top = 0;
            boundsRect.left = 0;
            boundsRect.bottom = 0;
            boundsRect.right = 144; /* 2 inches wide */
            mH = NewMeter(wPtr, &boundsRect, mTitle1,
                FALSE, 256L, 0L, 512L, 0L);
            (**mH).meterVis = TRUE;
            (**mH).meterHilite = 255;
            boundsRect.left = 150;
            boundsRect.right = 222; /* 1 inch wide */
            mH = NewMeter(wPtr, &boundsRect, mTitle2,
                FALSE, 0L, -1000L, 1000L, 0L);
            (**mH).meterVis = TRUE;
            (**mH).meterHilite = 255;
        }
        else if (which == 2) /* the front window at first */
        {
            boundsRect.top = 10;
            boundsRect.left = 50;
            boundsRect.bottom = 0;
            boundsRect.right = 158;

```

```

NewMeter(wPtr, &boundsRect, mTitle3, TRUE,
        2000L, 0L, 4095L, 0L);
boundsRect.left = 204;
boundsRect.right = 384;
NewMeter(wPtr, &boundsRect, mTitle4, TRUE,
        0L, -40950L, 40950L, 0L);
    }
} /* end MakeTwoMeters() */

/* Shake the needle around the setpoint to see it move.
Affects all meters in the front window. */
void DitherMeters()
{
    static int    random = 0;
    static long   oldVal = 0L;
    long         val, setVal, posIncrement, newVal;
    int          direction, docV, docH;
    ControlHandle chH, chV;
    MeterHandle  mH =
        (MeterHandle)((WindowPeek)thePort->refCon);

    if (ValidMeter(mH))
    {
        Delay(0L, &newVal);
        if (newVal - oldVal > 8L) /* time delay */
        {
            if (random > 0)
                direction = 1;
            else if (random < 0)
                direction = -1;
            else
                direction = 0;
            oldVal = newVal;
            /* get control handles */
            GetScrollHandles(thePort, &chH, &chV);
            /* find where we've scrolled to */
            docV = GetCtlValue(chV);
            docH = GetCtlValue(chH);
            /* shift to the right location in the document */
            SetOrigin(docH, docV);
            OffsetRgn(thePort->clipRgn, docH, docV);
            /* run through all the meters for this window */
            while (ValidMeter(mH))
            {
                val = (**mH).needleValue;
                setVal = (**mH).meterValue;
                posIncrement =
                    ((**mH).meterMax - (**mH).meterMin)/250L;
                newVal = val + (setVal - val)/10L + direction * posIn-
crement;
                SetNeedleValue(mH, newVal);
                mH = (**mH).nextMeter;
            }
            SetOrigin(0, 0);
            OffsetRgn(thePort->clipRgn, -docH, -docV);
            ++random;
            if (random >= 5)
                random = -5;
        }
    }
} /* end DitherMeters() */

/* Handle Mouse in window -can hit scroll bars or meters. */
void DoContent(theWindow, theEvent)
WindowPtr theWindow;
EventRecord*theEvent;
{
    int          cntlCode, axis;
    int          docV, docH;
    int          oldMWidth, newMWidth;
    Point        startPoint, endPoint;
    ControlHandle theCtl, chH, chV;
    MeterHandle  theMeter;
    Rect         limitRect, slopRect;

```

```

startPoint = theEvent->where;
GlobalToLocal(&startPoint);
/* First handle scroll bar events... */
if ((cntlCode =
    FindControl(startPoint, theWindow, &theCtl)) != 0)
{
    if (cntlCode >= inUpButton &&
        cntlCode <= inPageDown) /* not thumb */
        TrackControl(theCtl, startPoint, &ScrollProc);
    else if (cntlCode == inThumb)
    {
        /* Ordinarily you would call
        TrackControl(theCtl, startPoint, 0L) but today we're doing
        "real-time" thumb control. */
        ThumbControl(theCtl, startPoint);
    } /* else if in thumb */
} /* if in a control */
/*...then handle "true content" mouse events. */
else
{
    /* We're going to treat the meters as being
    attached to the underlying document rather than the window
    frame, so we must shift to "document" coordinates before
    determining which meter was hit. */
    LocalToDoc(&startPoint);
    if ((cntlCode =
        FindMeter(startPoint, theWindow, &theMeter)) != 0)
    {
        /* Shift everything to document coordinates
        before proceeding: */
        /* 1 - get control handles */
        GetScrollHandles(theWindow, &chH, &chV);
        /* 2 - find where we've scrolled to */
        docV = GetCtlValue(chV);
        docH = GetCtlValue(chH);
        /* 3 - shift to the right location in the document */
        SetOrigin(docH, docV);
        OffsetRgn(theWindow->clipRgn, docH, docV);
        /* 4 - now deal with events as though no scrolling has
        taken place */
        if (theEvent->modifiers & optionKey)
        {
            /* Drag the meter around. This is very handy
            since meters take up a lot of screen. Unneeded meters can be
            slid almost entirely off the screen to make room for other
            things. */
            slopRect = limitRect = thePort->portRect;
            InsetRect(&limitRect, 10, 10);
            limitRect.right -= SCROLLSIZE;
            limitRect.bottom -= SCROLLSIZE;
            axis = 0; /* no constraint */
            DragMeter(theMeter, startPoint,
                &limitRect, &slopRect, axis);
        }
        else if (theEvent->modifiers & cmdKey)
        {
            /* Resize the meter while mouse down - useful when
            playing with the look of the meter, and shows how the display
            box can be a very handy design tool. */
            oldMWidth = (**theMeter).meterRect.right
                - (**theMeter).meterRect.left;
            while (StillDown())
            {
                GetMouse(&endPoint); /* in unscrolled coord's */
                newMWidth = oldMWidth - startPoint.h + endPoint.h;
                SizeMeter(theMeter, newMWidth);
                DrawDisplayBox(theWindow, (void**)theMeter,
                    forMeter, 0, 0);
            }
        }
        else if (cntlCode >= inUpButton &&
            cntlCode <= inPageDown) /* not background */
            TrackMeter(theMeter, startPoint, &MeterProc);
    }
}

```

```

else if (cntlCode == inBackground)
{
    /* default for inBackGround just shows set
    point value */
    TrackMeter(theMeter, startPoint, 0L);
}
SetOrigin(0, 0);
OffsetRgn(theWindow->clipRgn, -docH, -docV);
} /* if FindMeter */
} /* else true content */
} /* end DoContent() */

/* The all-purpose redraw function, draws only what touches
the update region and handles scroll bars so that the routines
which draw individual items don't have to keep track of where
we've scrolled to in the document. Can even be used for
printing, provided you temporarily set the scroll bar values
to the minimum and the updateRgn to the whole document
beforehand, and provided your individual drawing routines
don't set the port before drawing. */
void UpdateContents(wPtr, frontMost)
WindowPtr wPtr;
Boolean frontMost;
{
    int docV, docH;
    Rect clipRect;
    ControlHandle cHH, cHV;
    MeterHandle mH = (MeterHandle)((WindowPeek)wPtr->refCon);

    SetClip(updateRgn);
    /* get control handles */
    GetScrollHandles(wPtr, &cHH, &cHV);
    /* find where we've scrolled to */
    docV = GetCtlValue(cHV);
    docH = GetCtlValue(cHH);
    /* shift to the right location in the document */
    if (docV || docH)
    {
        SetOrigin(docH, docV);
        OffsetRgn(wPtr->clipRgn, docH, docV);
    }
    /* we'll be ultra-efficient and draw only what
    touches the updateRgn */
    while (mH)
    {
        if (RectInRgn(&(**mH).meterRect, wPtr->clipRgn))
        {
            /* Fool ShowMeter() into redrawing the meter. */
            (**mH).meterVis = FALSE;
            ShowMeter(mH);
        }
        mH = (**mH).nextMeter;
    }
    if (docV || docH)
    {
        SetOrigin(0, 0);
        clipRect.top = wPtr->portRect.top;
        clipRect.left = wPtr->portRect.left;
        clipRect.bottom = wPtr->portRect.bottom - SCROLLSIZE;
        clipRect.right = wPtr->portRect.right - SCROLLSIZE;
        ClipRect(&clipRect);
    } /* end UpdateContents() */

    /* Show two integer values in a little display box at bottom
    right of the window. If you call this function with mode =
    forDebug, specify the WindowPtr and put the two values in
    newValue and otherValue. */
    void DrawDisplayBox(wPtr, theHandle, mode,
        newValue, otherValue)
    WindowPtr wPtr;
    void **theHandle;
    DisplayMode mode;
    int newValue, otherValue;
    {
        int i, j;

```

```

        int oldFont, oldSize;
        Style oldStyle;
        long leftNum, rightNum;
        char s[20], sr[10], *sPtr;
        Rect displayRect, clipRect;
        MeterHandle mH;
        ControlHandle theCtl, cHH, cHV;
        GrafPtr savePort;

        if (DISPLAYBOXWIDTH <= 0) return;
        GetPort(&savePort);
        SetPort(wPtr);
        oldFont = wPtr->txFont;
        oldSize = wPtr->txSize;
        oldStyle = wPtr->txFace; /* sic */
        TextFont(0);
        TextSize(12);
        TextFace(' ');
        displayRect.top = wPtr->portRect.bottom - SCROLLSIZE + 1;
        displayRect.left = wPtr->portRect.right
            - DISPLAYBOXWIDTH - 14;
        displayRect.bottom = wPtr->portRect.bottom + 1;
        displayRect.right = displayRect.left
            + DISPLAYBOXWIDTH - 1;
        ClipRect(&displayRect);
        EraseRect(&displayRect);

        if (mode == forWindow)
        {
            GetScrollHandles(wPtr, &cHH, &cHV);
            leftNum = (long)GetCtlValue(cHH);
            rightNum = (long)GetCtlValue(cHV);
            /* or try
            leftNum = wPtr->portRect.right - wPtr->portRect.left - 15;
            rightNum = wPtr->portRect.bottom - wPtr->portRect.top -
15;
            */
        }
        else if (mode == forMeter)
        {
            mH = (MeterHandle)theHandle;
            leftNum = (long)((**mH).meterRect.right
                - (**mH).meterRect.left);
            rightNum = (long)((**mH).meterRect.bottom
                - (**mH).meterRect.top);
        }
        else if (mode == forThumb)
        {
            GetScrollHandles(wPtr, &cHH, &cHV);
            theCtl = (ControlHandle)theHandle;
            if (theCtl == cHH)
            {
                leftNum = (long)newValue;
                rightNum = (long)GetCtlValue(cHV);
            }
            else
            {
                leftNum = (long)GetCtlValue(cHH);
                rightNum = (long)newValue;
            }
        }
        else if (mode == forDebug)
        {
            leftNum = (long)newValue;
            rightNum = (long)otherValue;
        }
        else
        {
            leftNum = 0L;
            rightNum = 0L;
        }
        NumToString(leftNum, s);
        NumToString(rightNum, sr);
        /* Concatenate sr into s with space between. */

```

```

j = s[0];
s[++j] = ' ';
for (i = 1, sPtr = &(s[j + 1]); i <= sr[0]; ++i)
    *sPtr++ = sr[i];
s[0] += i; /* note 1 extra for space */
TextBox(&(s[1]), (Byte)(s[0]), &displayRect,
        teJustCenter);

clipRect.top = thePort->portRect.top;
clipRect.left = thePort->portRect.left;
clipRect.bottom = thePort->portRect.bottom - SCROLLSIZE;
clipRect.right = thePort->portRect.right - SCROLLSIZE;
ClipRect(&clipRect);
TextFont(coldFont);
TextSize(coldSize);
TextFace(coldStyle);
SetPort(savePort);
} /* end DrawDisplayBox() */

/* Both DoZoomWindow() and DoGrowWindow() handle the poten-
tially embarrassing problem of moving around in a document of
fixed size without scrolling out into the Twilight Zone. */
void DoZoomWindow (wPtr, part)
WindowPtr wPtr;
int part;
{
    int oldV, oldH, newV, newH;
    int vertRange, horRange;
    int curValH, curMaxH, curValV, curMaxV;
    Point dS;
    ControlHandle cHH, cHV;

    /* get control handles */
    GetScrollHandles(wPtr, &cHH, &cHV);
    oldV = (wPtr->portRect).bottom - (wPtr->portRect).top;
    oldH = (wPtr->portRect).right - (wPtr->portRect).left;
    curValH = GetCtlValue(cHH);
    curMaxH = GetCtlMax(cHH);
    curValV = GetCtlValue(cHV);
    curMaxV = GetCtlMax(cHV);
    ZoomWindow(wPtr, part, TRUE);
    InvalRect(&(wPtr->portRect));
    newV = (wPtr->portRect).bottom - (wPtr->portRect).top;
    newH = (wPtr->portRect).right - (wPtr->portRect).left;
    SetRect(&((**cHV).controlRect),
            wPtr->portRect.right - SCROLLSIZE,
            wPtr->portRect.top - 1, wPtr->portRect.right + 1,
            wPtr->portRect.bottom - 14);
    SetRect(&((**cHH).controlRect), wPtr->portRect.left - 1,
            wPtr->portRect.bottom - SCROLLSIZE,
            wPtr->portRect.right - DISPLAYBOXWIDTH - 14,
            wPtr->portRect.bottom + 1);
    vertRange = MAXWINDOWHEIGHT -
        (wPtr->portRect.bottom - wPtr->portRect.top);
    vertRange = (vertRange > 0) ? vertRange : 0;
    horRange = MAXWINDOWWIDTH -
        (wPtr->portRect.right - wPtr->portRect.left);
    horRange = (horRange > 0) ? horRange : 0;
    SetCtlMax(cHV, vertRange);
    SetCtlMax(cHH, horRange);
    if (vertRange == 0)
        HiliteControl(cHV, 255);
    else
        HiliteControl(cHV, 0);
    if (horRange == 0)
        HiliteControl(cHH, 255);
    else
        HiliteControl(cHH, 0);
    /* "Scroll" if running off the bottom or right. The whole
    window has been invalidated above, so we just shift scroll
    bars & then let update event take care of redrawing later. */
    /* dS.h(or .v) = overshoot = amount potentially exposed
    by zoom less amount available to be exposed; potential amount
    = increase in window size; amount available = maximum amount

```

```

left unseen, which is the (prezoom) control Max, less amount
that has already scrolled by, which is the (prezoom) control
Value. If you grok that, you've got scroll bars in the bag! */
dS.h = newH - oldH - curMaxH + curValH;
dS.h = (dS.h > 0) ? dS.h : 0;
dS.v = newV - oldV - curMaxV + curValV;
dS.v = (dS.v > 0) ? dS.v : 0;
if (dS.h > 0 || dS.v > 0)
{
    SetCtlValue(cHV, GetCtlValue(cHV) - dS.v);
    SetCtlValue(cHH, GetCtlValue(cHH) - dS.h);
}
} /* end DoZoomWindow() */

void DoGrowWindow(Event)
EventRecord *event;
{
    int winWidth, winHeight;
    int horRange, vertRange;
    int oldV, oldH, newV, newH;
    int curValH, curMaxH, curValV, curMaxV;
    long growResult;
    Point dS;
    Rect limitRect, badRect;
    ControlHandle cHH, cHV;
    RgnHandle tempRgn;
    WindowPtr wPtr = FrontWindow();

    winHeight = ((screenBits.bounds.bottom
        - screenBits.bounds.top - 22)
        <= MAXWINDOWHEIGHT) ? (screenBits.bounds.bottom
        - screenBits.bounds.top - 22) : MAXWINDOWHEIGHT;
    winWidth = ((screenBits.bounds.right
        - screenBits.bounds.left - 2)
        <= MAXWINDOWWIDTH) ? (screenBits.bounds.right
        - screenBits.bounds.left - 2) : MAXWINDOWWIDTH;

    limitRect.top = 130;
    limitRect.left = 250;
    limitRect.bottom = winHeight + 1;
    limitRect.right = winWidth + 1;
    /* get control handles */
    GetScrollHandles(wPtr, &cHH, &cHV);
    oldV = (wPtr->portRect).bottom - (wPtr->portRect).top;
    oldH = (wPtr->portRect).right - (wPtr->portRect).left;
    curValH = GetCtlValue(cHH);
    curMaxH = GetCtlMax(cHH);
    curValV = GetCtlValue(cHV);
    curMaxV = GetCtlMax(cHV);
    growResult = GrowWindow(wPtr, event->where, &limitRect);
    newV = HiWord(growResult);
    newH = LoWord(growResult);
    SizeWindow(wPtr, newH, newV, TRUE);
    ClipRect(&(wPtr->portRect));
    SetRect(&((**cHV).controlRect),
            wPtr->portRect.right - SCROLLSIZE,
            wPtr->portRect.top - 1,
            wPtr->portRect.right + 1,
            wPtr->portRect.bottom - 14);
    SetRect(&((**cHH).controlRect), wPtr->portRect.left - 1,
            wPtr->portRect.bottom - SCROLLSIZE,
            wPtr->portRect.right - DISPLAYBOXWIDTH - 14,
            wPtr->portRect.bottom + 1);
    vertRange = MAXWINDOWHEIGHT -
        (wPtr->portRect.bottom - wPtr->portRect.top);
    vertRange = (vertRange > 0) ? vertRange : 0;
    horRange = MAXWINDOWWIDTH -
        (wPtr->portRect.right - wPtr->portRect.left);
    horRange = (horRange > 0) ? horRange : 0;
    SetCtlMax(cHV, vertRange);
    SetCtlMax(cHH, horRange);
    if (vertRange == 0)
        HiliteControl(cHV, 255);
    else
        HiliteControl(cHV, 0);

```

```

if (horRange == 0)
    HiliteControl(cHH, 255);
else
    HiliteControl(cHH, 0);
/* update scroll bars properly */
if (newV > oldV)
{
    badRect.top = oldV - SCROLLSIZE;
    badRect.left = 0;
    badRect.bottom = oldV;
    badRect.right = oldH;
    InvalRect(&badRect);
}
else
{
    badRect.top = (wPtr->portRect).bottom - SCROLLSIZE;
    badRect.left = (wPtr->portRect).left;
    badRect.bottom = (wPtr->portRect).bottom;
    badRect.right = (wPtr->portRect).right;
    InvalRect(&badRect);
}
if (newH > oldH)
{
    badRect.top = 0;
    badRect.left = oldH - SCROLLSIZE;
    badRect.bottom = oldV;
    badRect.right = oldH;
    InvalRect(&badRect);
}
else
{
    badRect.top = (wPtr->portRect).top;
    badRect.left = (wPtr->portRect).right - SCROLLSIZE;
    badRect.bottom = (wPtr->portRect).bottom;
    badRect.right = (wPtr->portRect).right;
    InvalRect(&badRect);
}
badRect.top = (wPtr->portRect).top;
badRect.left = (wPtr->portRect).left;
badRect.bottom = (wPtr->portRect).bottom - SCROLLSIZE;
badRect.right = (wPtr->portRect).right - SCROLLSIZE;
/* Scroll if running off the bottom or right of document.
The newly exposed part is invalidated. The resulting update
event will set updateRgn to the total invalidated region,
and UpdateContents() will redraw only that part of window. */
dS.h = newH - oldH - curMaxH + curValH;
dS.h = (dS.h > 0) ? dS.h : 0;
dS.v = newV - oldV - curMaxV + curValV;
dS.v = (dS.v > 0) ? dS.v : 0;
if (dS.h > 0 || dS.v > 0)
{
    ClipRect(&badRect);
    tempRgn = NewRgn();
    SetCtlValue(cHV, GetCtlValue(cHV) - dS.v);
    SetCtlValue(cHH, GetCtlValue(cHH) - dS.h);
    ScrollRect(&badRect, dS.h, dS.v, tempRgn);
    InvalRgn(tempRgn);
    DisposeRgn(tempRgn);
}
else
    ClipRect(&badRect);
} /* end DoGrowWindow() */

void DoCloseWindow()
{
    WindowPtr wPtr = FrontWindow();

    KillMeters(wPtr);
    CloseWindow(wPtr); /* does KillControls() for you */
} /* end DoCloseWindow() */

Boolean AMeterWindow(wPtr)
WindowPtr wPtr;
{

```

```

if (wPtr == (WindowPtr)(&wRecord1) ||
    wPtr == (WindowPtr)(&wRecord2))
    return(TRUE);
return(FALSE);
} /* end AMeterWindow() */

```

```

void GetScrollHandles(wPtr, ch, cv)
WindowPtr wPtr;
ControlHandle *cv, *ch;
{
    ControlHandle cHH, cHV;

    cHH = ((WindowPeek)wPtr)->controlList;
    while (cHH)
    {
        if (GetCRefCon(cHH) == HSCROLL) break;
        cHH = (**cHH).nextControl;
    }
    cHV = ((WindowPeek)wPtr)->controlList;
    while (cHV)
    {
        if (GetCRefCon(cHV) == VSCROLL) break;
        cHV = (**cHV).nextControl;
    }
    *ch = cHH;
    *cv = cHV;
} /* end AMeterWindow() */

```

```

void LocalToDoc(thePt)
Point *thePt;
{
    ControlHandle cHH, cHV;

    /* get control handles */
    GetScrollHandles(thePort, &cHH, &cHV);
    /* Document coordinates are always at least as large as
local coord's - a mental picture: the window sliding over the
document helps. */
    thePt->v += GetCtlValue(cHV);
    thePt->h += GetCtlValue(cHH);
} /* end LocalToDoc() */

```

```

void MeterProc(mH, theCode)
MeterHandle mH;
int theCode;
{
    long posIncrement = ((**mH).meterMax
                        - (**mH).meterMin)/250L;

    long scrollAmt;

    Delay(2L, &scrollAmt);
    switch (theCode)
    {
        case inUpButton:
            scrollAmt = posIncrement/10L;
            if (!scrollAmt)
                scrollAmt = 1L;
            break;
        case inDownButton:
            scrollAmt = -posIncrement/10L;
            if (!scrollAmt)
                scrollAmt = -1L;
            break;
        case inPageUp:
            scrollAmt = posIncrement;
            break;
        case inPageDown:
            scrollAmt = -posIncrement;
            break;
    }
    SetMtrValue(mH, GetMtrValue(mH) + scrollAmt);
} /* end MeterProc() */

```

```

pascal void ScrollProc(theCtl, partCode)

```

```

ControlHandle theCtl;
int partCode;
{
    int delta, pageSize, arrowSize, oldValue;
    Point dS;
    Rect viewBnds;
    Boolean horizontal = FALSE;

    /* which control, how far */
    if (GetCRefCon(theCtl) == HSCROLL)
    {
        horizontal = TRUE;
        pageSize = (thePort->portRect).right
            - (thePort->portRect).left - SCROLLSIZE;
    }
    else /* VSCROLL */
        pageSize = (thePort->portRect).bottom
            - (thePort->portRect).top - SCROLLSIZE;
    arrowSize = pageSize/20;
    /* delta is used to adjust local coord's, which DECREASE
    if up button pressed (we move closer to top of document).
    However, we want the stuff in the window to slide DOWN so the
    sign of dS is reversed from that of delta in order that
    ScrollRect() will work correctly. Hence for up button of
    vertical scroll, delta is negative and dS.v is positive (they
    agree in magnitude unless we're at one end or the other of the
    document).*/
    switch (partCode)
    {
        case inUpButton:
            delta = -arrowSize;
            break;
        case inDownButton:
            delta = arrowSize;
            break;
        case inPageUp:
            delta = -pageSize;
            break;
        case inPageDown:
            delta = pageSize;
            break;
        default:
            delta = 0;
            break;
    } /* end switch */
    ClipRect(&(thePort->portRect));
    oldValue = GetCtlValue(theCtl);
    SetCtlValue(theCtl, oldValue + delta);
    if (horizontal)
    {
        dS.h = oldValue - GetCtlValue(theCtl);
        dS.v = 0;
    }
    else /* vertical */
    {
        dS.v = oldValue - GetCtlValue(theCtl);
        dS.h = 0;
    }
    viewBnds.top = (thePort->portRect).top;
    viewBnds.left = (thePort->portRect).left;
    viewBnds.bottom = (thePort->portRect).bottom
        - SCROLLSIZE;
    viewBnds.right = (thePort->portRect).right
        - SCROLLSIZE;

    /* To speed redrawing up, you could just
    set updateRgn equal to viewBnds and call UpdateContents()
    without scrolling if the partCode was for paging- however this
    gives the user less visual feedback. */
    /* ScrollRect() both moves the screen image and accumulates
    the freshly exposed region in updateRgn. */
    ScrollRect(&viewBnds, dS.h, dS.v, updateRgn);
    DrawDisplayBox(thePort, 0L, forWindow, 0, 0);
    UpdateContents(thePort, TRUE);
} /* end ScrollProc() */

```

```

/* This function will track the mouse and correctly update the
value of a scroll bar as the thumb is dragged around, in case
you might want to update a graphic or number continuously as
the user moves the thumb about, something TrackControl()
doesn't readily allow. Use "ThumbControl(startPoint, theCtl);"
in place of "if (TrackControl(theCtl, startPoint, 0L))" when
FindControl() reports an appropriate thumb event, and modify
the action calls in this routine where indicated. Depending on
your needs, you may want to clip differently. For generality,
this routine decides which scroll bar is which by the same
method that the Control Manager uses - which way is longer?
Much of "while (StillDown())" below is a duplication of
DragGrayRgn() that lets us tell where we are at all times. */
void ThumbControl(theCtl, startPoint)
ControlHandle theCtl;
Point startPoint;
{
    int newValue, startValue, thickness;
    int mouseRange, controlMin, controlMax;
    long thumbRange, controlRange;
    Point theMouse, thumbStart, thumbMin,
        thumbMax, newThumbCent, thumbCent, halfThumb;
    Rect controlRect, slopRect, startRect, thumbRect;
    RgnHandle oldClipRgn;
    PenState pState;
    Boolean isHorizontalBar, tracking;

    GetPenState(&pState);
    PenMode(patXor);
    PenPat(dkGray);
    PenSize(1, 1);
    controlRect = (*theCtl).controlRect;
    slopRect = controlRect;
    controlMin = GetCtlMin(theCtl);
    controlMax = GetCtlMax(theCtl);
    controlRange = (long)(controlMax - controlMin);
    startValue = GetCtlValue(theCtl);
    newValue = startValue;
    isHorizontalBar =
        (controlRect.right - controlRect.left) >
        (controlRect.bottom - controlRect.top) ?
        TRUE : FALSE;
    if (isHorizontalBar)
    {
        thumbStart.v = controlRect.top +
            (controlRect.bottom - controlRect.top + 1)/2;
        thumbMin.v = thumbMax.v = thumbStart.v;
        thickness = controlRect.bottom - controlRect.top;
        halfThumb.h = thickness/2;
        halfThumb.v = halfThumb.h - 1;
        thumbMin.h = controlRect.left
            + thickness + (thickness + 1)/2;
        thumbMax.h = controlRect.right
            - thickness - (thickness + 1)/2;
        thumbRange = (long)(controlRect.right
            - controlRect.left - 3*thickness);
        thumbStart.h = thumbMin.h
            + (thumbRange*(startValue - controlMin)
            + controlRange/2)/controlRange;
        InsetRect(&slopRect, -72, -36);
    }
    else
    {
        thumbStart.h = controlRect.left +
            (controlRect.right - controlRect.left + 1)/2;
        thumbMin.h = thumbMax.h = thumbStart.h;
        thickness = controlRect.right - controlRect.left;
        halfThumb.v = thickness/2;
        halfThumb.h = halfThumb.v - 1;
        thumbMin.v = controlRect.top
            + thickness + (thickness + 1)/2;
        thumbMax.v = controlRect.bottom
            - thickness - (thickness + 1)/2;
    }
}

```



```

thumbRange = (long)(controlRect.bottom
- controlRect.top - 3*thickness);
thumbStart.v = thumbMin.v
+ (thumbRange*(startValue - controlMin)
+ controlRange/2)/controlRange;
InsetRect(&slopRect, -36, -72);
}
if (thumbRange <= 0) return; /* that's a pretty
short scroll bar */
newThumbCent = thumbCent = thumbStart;
oldClipRgn = NewRgn();
GetClip(oldClipRgn);
ClipRect(&controlRect);
SetRect(&thumbRect, thumbCent.h - halfThumb.h,
thumbCent.v - halfThumb.v, thumbCent.h + halfThumb.h,
thumbCent.v + halfThumb.v);

tracking = FALSE;
while (StillDown())
{
ClipRect(&controlRect);
GetMouse(&theMouse);
if (PtInRect(theMouse, &slopRect))
{ /* mouse is close to scroll bar - track the thumb */
/* calculate new thumb center */
if (isHorizontalBar)
{
newThumbCent.h = thumbStart.h
+ (theMouse.h - startPoint.h);
if (newThumbCent.h < thumbMin.h)
newThumbCent.h = thumbMin.h;
else if (newThumbCent.h > thumbMax.h)
newThumbCent.h = thumbMax.h;
}
else
{
newThumbCent.v = thumbStart.v
+ (theMouse.v - startPoint.v);
if (newThumbCent.v < thumbMin.v)
newThumbCent.v = thumbMin.v;
else if (newThumbCent.v > thumbMax.v)
newThumbCent.v = thumbMax.v;
}
}
if (tracking)
{
if (newThumbCent.h != thumbCent.h ||
newThumbCent.v != thumbCent.v)
{
FrameRect(&thumbRect);
SetRect(&thumbRect, newThumbCent.h - halfThumb.h,
newThumbCent.v - halfThumb.v,
newThumbCent.h + halfThumb.h,
newThumbCent.v + halfThumb.v);
FrameRect(&thumbRect);
}
}
else /* just entering slop rect */
{
SetRect(&thumbRect,
newThumbCent.h - halfThumb.h,
newThumbCent.v - halfThumb.v,
newThumbCent.h + halfThumb.h,
newThumbCent.v + halfThumb.v);
FrameRect(&thumbRect);
tracking = TRUE;
}
} /* if track the thumb */
else /* not in slop rect */
{
if (tracking)
{ /* just leaving slop rect */
FrameRect(&thumbRect);
tracking = FALSE;
newThumbCent = thumbStart;
}
}
}

```

```

} /* else default to start position */
if (newThumbCent.h != thumbCent.h ||
newThumbCent.v != thumbCent.v)
{
/* Either thumb has been dragged to new position or
mouse has left slopRect, in which case thumbCent reverts to
thumbStart. */
/* Calculate control value based on newThumbCent. */
newValue = (controlRange*(newThumbCent.h
- thumbMin.h + newThumbCent.v - thumbMin.v)
+ thumbRange/2)/thumbRange + controlMin;
thumbCent = newThumbCent;
}
/* Insert your custom call here - the current control value is
"newValue"; reset clipRgn if you draw outside scroll bar! */
/* Sample thumb action; echo control values to display box. */
DrawDisplayBox(thePort, (void**)theCtl, forThumb, newValue, 0);
}
} /* while still down */
if (tracking)
FrameRect(&thumbRect);
if (newValue != startValue)
{
SetCtlValue(theCtl, newValue);
/* The control value has changed on mouseup to "newValue"
- take appropriate action - here we redraw the whole window,
but if newValue were close to startValue you could call
ScrollRect() instead. */
InvalRect(&(thePort->portRect));
}
else
{
/* control value hasn't changed - reset if necessary */;
SetClip(oldClipRgn);
DisposeRgn(oldClipRgn);
SetPenState(&pState);
} /* end ThumbControl() */

void PutMetersOnClip(wPtr)
WindowPtr wPtr;
{
Rect theClipRect;
PicHandle clipPicH;
RgnHandle oldClipRgn;
MeterHandle mH = (MeterHandle)((WindowPeek)wPtr->refCon);

ZeroScrap();
theClipRect.top = 0;
theClipRect.left = 0;
theClipRect.bottom = DOHEIGHT;
theClipRect.right = DOWIDTH;
oldClipRgn = NewRgn();
GetClip(oldClipRgn);
ClipRect(&theClipRect);

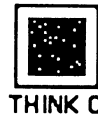
/* draw the picture */
clipPicH = OpenPicture(&theClipRect);
while (ValidMeter(mH))
{
MeterSnapshot(mH);
mH = (**mH).nextMeter;
}
ClosePicture();
/* put it on the scrap and clean up */
HLock(clipPicH);
PutScrap(GetHandleSize(clipPicH), 'PICT', *clipPicH);
HUnlock(clipPicH);
KillPicture(clipPicH);
SetClip(oldClipRgn);
DisposeRgn(oldClipRgn);
} /* end PutMetersOnClip() */
/* End MeterMain.c */

```



C Workshop

Alternate Scroll Bar CDEF



Alexander S. Colwell
Redondo Beach, CA
Volume 5, Number 11

Introduction

Why another CDEF? MacTutor already have several articles relating to the CDEF. Especially relating to is the excellent *Designer CDEFs* article in April 1989 issue. Having read every single MacTutor issue, I've never seen a scroll bar CDEF article containing sample code!

I wanted a scroll bar that has two up and down arrow buttons on both ends of the scroll bar as shown in Figure 1, rather than standard single arrow buttons scroll bar. My primary reason for a dual arrow buttons scroll bar is for large 19-inch screen monitors. I find it's a real pain to perform simple scrolling function whenever your window is fully zoomed-out. I must pick-up the mouse several times to get the mouse cursor position over the scroll bar's arrow buttons. Or, I need a large uncluttered desk to navigate the mouse cursor over the scroll bar. Anyway, enough excuses!

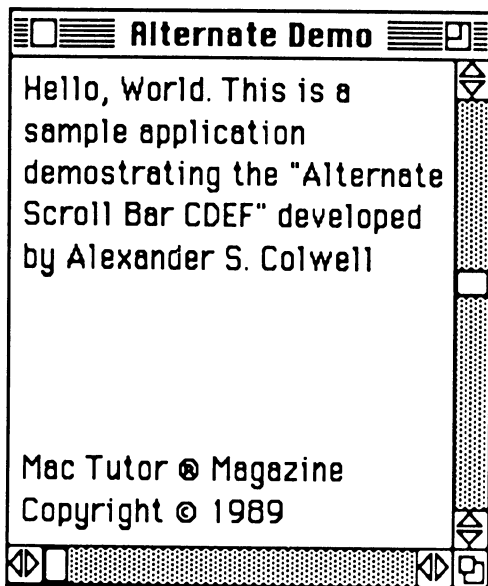


Figure 1

CDEF Definition

The scroll bar control definition comprises two sections: control's messages and control's color. The messages are described in Inside Macintosh Volume I in Control Manager chapter. I found I had to read the CDEF description in the Control Manager chapter several times before I fully understood it's implications on certain issues relating to message relationships with the ROM routines. The control's color is described in the Inside Macintosh Volume V in Control Manager chapter. (I only had to read this chapter a couple of times relating to the CDEF's

color controls. Either I am getting better or the ROM documentation is getting better. I'm more likely inclined to believe the latter).

Each of the control messages are described relating to the scroll bar definition. I will not reiterate the same description found in the Control Manager chapter, but I will attempt to describe how the scroll bar definition is related to each of the control messages.

Scroll Bar Layout

Figure 2 shows how the alternate scroll bar layout would appear in the user's application or desk accessory. This is similar to the Control Manager chapter in Inside Macintosh Volume I description of the standard scroll bar's parts. The functionality between the standard scroll bar and this alternate scroll bar is identical. The only differences are the appearance of the "up arrow" and "down arrow" control parts.

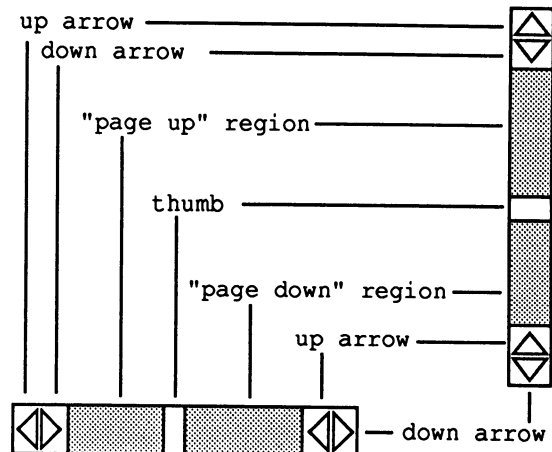


Figure 2

Entry Point

```
pascal long main(varCode, theControl,
message, param)
short varCode; /* Variation ctrl code */
ControlHandle theControl; /* Ctrl hdl */
short message; /* Ctrl message code */
long param; /* Ctrl parameter code */
```

The scroll bar entry point receives four arguments: *varCode*, *theControl*, *message*, and *param*. This scroll bar definition does not have any variation. Therefore the *varCode* argument is never used. The "simple button", "check box", and "radio button" controls are examples of the variation control code

usage.

The *theControl* variable is the scroll bar's control handle. This is where the definition keeps all its necessary data structures for various control's operations.

The *message* argument informs the scroll bar to perform the necessary control operations. This definition will use all the control messages except *autoTrack* control message. Each message will be fully described in the following sections.

The *param* argument is extra information passed with the associated control message. The scroll bar definition code is type-casted the *param*'s value before using it. For example, the *testCntl* control message's *param* value is "Point" data type while the *calcCRgns* control message's *param* value is "RgnHandle" data type. The usage of the *param*'s value is described in each of the control message that is used by this scroll bar definition.

drawCntl

The *drawCntl* control message comprises the largest chunk of code and the most complex portion of the scroll bar definition. The different parts of the scroll bar to be drawn depends on several combination of the CDEF's *param*, and *ControlRecord*'s *ctrlVis*, *ctrlValue/ctrlMin/ctrlMax*, and *ctrlHilite* variables.

The CDEF's *param* variable contains different values for different parts of the scroll bar operations. Note: The *param* value is type-casted as *long* word. The actual value is 16-bit low word quantity. This is not clear in the Inside Macintosh I Control Manager chapter but is explained in the Technical Note #196. Each *param* is described in the following:

param = 0: This indicates that the scroll bar should be completely re-drawn during window's update or control's transition from an invisible to visible state.

param = 0xff: This indicates that the scroll bar should transition into unhighlighted state. The scroll bar's "page up" and "page down" and "thumb" parts should be removed from the scroll bar.

param = *inPageUp*: This indicates that the "page up" region should be re-drawn.

param = *inPageDown*: This indicates that the "page down" region should be re-drawn.

param = *inUpButton*: This indicates that the either upper or lower "up arrow" should be highlighted or unhighlighted. If the *ctrlHilite* value is equal to zero then the "up arrow" should be unhighlighted. Otherwise, if the *ctrlHilite* value is equal to non-zero, and if the *ctrlMin* and *ctrlMax* values are not the same value, then it should highlight the "up arrow" button. Note: The *altButton* variable in the CDEF is used to determined if using the upper or lower arrows for highlighting or unhighlighting purposes.

param = *inDownButton*: This indicates that the either upper or lower "down arrow" should be highlighted or unhighlighted. The highlighting rules are the same as the *param*'s *inUpButton* case.

testCntl

The *testCntl* message should check if the scroll bar control is currently visible then it checks if the mouse is within the

control's *ctrlRect* rectangle area by comparing the mouse location specified by the *param*'s "Point" value.

If the mouse location is within the control's rectangle area then *DSTestControl* function is invoked to compare the mouse location with the different parts of the scroll bar : *inPageUp*, *inPageDown*, *inThumb*, both *inUpButtons*, and both *inDownButtons*. If a matching control part is found then it will return the scroll bar's control part. Otherwise, zero is returned indicating no scroll bar's part was found. Note: Again, the *altButton* variable is set depending on which of the upper and lower arrow buttons selected for highlighting and unhighlighting purposes.

calcCRgns

The *calcCRgns* control message should return the specified scroll bar part's region. The *param*'s "RgnHandle" value contains a region handle to be reset to the scroll bar part's region. However, there are only two cases this control message is invoked.

First case, the whole scroll bar region indicated by the high-order bit of *param* is set. Note: The *param* region's high-order bit should be cleared before operating on the region handle. Otherwise, the CDEF runs a risk of ALA BOMBA country. Technically speaking, the *param* value should not have high-order bit set in the region handle due to 32-bit mode addressing future implementation. But according to Technical Note #212 the solution for this particular problem is to separate the two cases into *calcCntlRgn* and *calcThumbRgn* control messages equivalent to this message for System 7.0 and henceforth versions in 32-bit mode addressing.

The second case is the "thumb" scroll bar part request by *TrackControl* ROM function. The *TrackControl* ROM function will invoke the *calcCRgns* message for the "thumb" outline area to be passed onto the *DragGrayRgn* ROM function for dragging the "thumb" part of the scroll bar. One extra thing this message should do is set the drag region's pattern used by the *DragGrayRgn* routine. I set *liGray* pattern into the global *DragPattern* variable. This is the same pattern used to fill-in the *inPageUp* and *inPageDown* control parts.

calcCntlRgn

The *calcCntlRgn* control message is equivalent to the *calcCRgns*'s first case.

calcThumbRgn

The *calcThumbRgn* control message is equivalent to the *calcCRgns*'s second case.

initCntl

The *initCntl* control message is where any necessary initialization required by the scroll bar control. Typically, the control definition will allocate one or more data structures to be used during the course of the control's operations.

This scroll bar will allocate DS handle to be used for the scroll bar's operation saved into the *ctrlData* variable of the *ControlRecord*. During the initialization, the black and light gray pattern are initialized. The light gray pattern is used in the

“upPage” and “downPage” regions of the scroll bar, and for the “thumb” drag region area. The DS’s hasColor variable is initialized indicating if the monitor has color by using the SysEnvi-
rons’s hasColorQD variable. The scroll bar definition uses this variable to determine if will need to perform color drawing of the scroll bar parts. Finally, the *cntlAction* variable is initialized to -1L for the TrackControl’s actionProc parameter “thumb” tracking.

The DSInitControl routine will be invoked to initialize the DS data structure. It’s primary job is to compute the size of the “thumb” rectangle area and create the polygon handle equivalent to the “up arrow” and “down arrow” buttons.

dispCntl

The *dispCntl* control message informs the scroll bar control to release all it’s internal data structures within the *controlData* handle. This scroll bar will release the polygon handles and the DS handle to the memory manager.

posCntl

The *posCntl* control message informs the scroll bar definition to re-compute the new “thumb” position and re-draw the scroll bar with the new “thumb” position. The *param*’s “Point” value contains the local-coordinates point position within the scroll bar’s *cntlRect* rectangle area. The scroll bar definition will look at the *h* part of the point for horizontal scroll bar, and use *v* part of the point for vertical scroll bar orientation.

thumbCntl

The *thumbCntl* control message informs the scroll bar definition to compute the “thumb” dragging rectangle area. The *param* value contains a pointer to a data structure described in DragGrayRgn in Window Manager chapter. It will briefly describe it here:

```
struct (
    Rect limitRect; /* Dragging region*/
    Rect limitRect; /* Limit rect scr1*/
    Rect slopRect; /* Slop rect scr1 */
    short axis; /* Draw axis ctrl */
);
```

The scroll bar definition will set the *axis* variable to *hAxisOnly* constant for horizontal scroll bar or *vAxisOnly* constant for vertical scroll bar. The *limitRect* variable is the rectangle area comprising the “upPage” and “downPage” rectangle areas. And, the *slopRect* variable is the slop area allowed for the mouse to move outside the limit rectangle area.

The *slopRect* rectangle area require special calculation base on the current mouse position within the “thumb”. Before setting up the limit rectangle area, the *limitRect.left* and *limitRect.top* contain the current mouse position. This is saved into temporary mouse point variable to be used in the slop rectangle area calculation. The problem is that the slop rectangle area cannot be the same as the limit rectangle area. Or else, the “thumb” dragging outline will go over the “up arrow” and “down arrow” regions. Hence, the slop rectangle area must be adjusted where the mouse position within the “thumb” rectangle area.

dragCntl

The *dragCntl* control message informs the scroll bar that the control or the “thumb” is going to be dragged since, the scroll bar has no custom dragging requirements. It returns zero value to inform the Control Manager to use the default dragging method.

autoTrack

The *autoTrack* control message is not used by the scroll bar definition. Since the *initCntl* control message initializes the ControlRecord’s *cntlAction* to -1L value. This will cause the TrackControl call the default control tracking mechanism. Since the *dragCntl* message always return zero value this control message is never invoked. Note: I had always been confused on the purpose of the *dragCntl* and *autoTrack* control message combinations. Perhaps, someone could explain this in Apple’s Technical Note series someday.

Color descriptions

The scroll bar CDEF uses the GetAuxCtl ROM routine to determine how to color different parts of the scroll bar. The CDEF uses four different RGB colors for the control’s frame, body, text, and thumb parts.

The control’s frame color is the outline parts of the scroll bar, arrows, up & down page regions, and thumb. The control’s body color is the inside parts of the arrows, and up & down page regions. The control’s thumb color is the inside part of the thumb. The text color is not used in this scroll bar definition. Note: Standard control buttons uses the control’s text color value.

Accessing the scroll bar’s colors is very easy. The GetAuxCtl ROM call is used to retrieve the scroll bar’s colors. The calling proto-type definition is as follows:

```
Boolean
GetAuxCtl(ControlHandle theControl,
    AuxCtlHndl &acHndl);
```

The function will return TRUE indicator if the scroll bar’s *theControl* control handle has it’s own copy of the color data set. Otherwise, FALSE indicates the scroll bar using default color data set. The *acHndl* auxiliary control handle will return the scroll bar’s color data set. To access the frame, body, and thumb RGB colors are shown in the following:

```
(*(*acHndl)->acCTable)->
    ctTable[cFrameColor].rgb

(*(*acHndl)->acCTable)->
    ctTable[cBodyColor].rgb

(*(*acHndl)->acCTable)->
    ctTable[cThumbColor].rgb
```

I recommend reading Inside Macintosh V’s Control Manager chapter on more details relating to the control’s color displays. There are two main color routines relating to the controls : GetAuxCtl and SetCtlColor.

Other CDEF Descriptions

The scroll bar definition uses a weird technique handling black & white and color situations. I defined DSDrawObject to draw various control's part onto the window. This generic routine simplifies the scroll bar definition without worrying if drawing black & white or color. And, the DSDrawObject routine is coupled with series of macro definitions to perform various drawing operations for the scroll bar.

Finally, I used floating-point to perform various calculations to position the "thumb" properly based on the control's *ctrlValue*, *ctrlMin*, and *ctrlMax* variables. I could have used fix-integer arithmetic, but I was too lazy to bother with it.

Scroll Bar Demonstration

The scroll bar demonstration is a simple text editor application using new THINK C 4.0 Object Library. It's not great American word processor, but it is used to illustrate the Alternate Scroll Bar CDEF.

Wrapping Things Up

I did not realize how difficult writing a scroll bar definition was until writing this CDEF example. I am not for sure this is a good alternative scroll bar interface, but it was fun writing it. I believe Apple's Human Interface Guidelines are excellent set of rules to follow. So how does this scroll bar definition fit into their scheme of things? Probably none. However, several of my associates at work want a scroll bar definition that varies the size of the "thumb" relative to how much visible text (or contents) versus hidden text, but I said "It's almost impossible, since the CDEF does not have enough information to determine how to vary the 'thumb' size.". Anyway, until then onward to other Mac programming!

```
/*
 *
 * Source - Alternate.c
 * Author - Alexander S. Colwell,
 * Copyright (C) 1989
 * Purpose - This application will demonstrate the
 * AlternateCDEF scroll bar using THINK
 * C 4.0 Object Library.
 */

#include <Global.h> /* Global variables */
#include <Application.h> /* Application defs */
#include <Commands.h> /* Command handler defs */
#include <CBartender.h> /* Menu handler defs */
#include <CDecorator.h> /* Window handler defs */
#include <CDesktop.h> /* Window layer hdlr def*/
#include <CDocument.h> /* Document handler defs */
#include <CEditText.h> /* Text edit handler defs*/
#include <CScrollPane.h> /* Scrolling pane defs */

struct CAltDemoApp : CApplication { /* Appl object */
void CreateDocument(void); /* Creat new doc method */
};

struct CAltDemoDoc : CDocument { /* Document object */
void NewFile(void); /* Net (bogus) fil method*/
};

struct CAltDemoPane : CEditText { /* Text pane obj */
```

```
void IEditPane(CView *anEnclosure,
CBureaucrat *aSupervisor);
};

#define altWIND 500 /* WIND template res ID */

/* Define external refs */
extern CApplication *gApplication; /* Appl object */
extern CDecorator *gDecorator; /* Window object */
extern CDesktop *gDesktop; /* Desktop object */
extern CBartender *gBartender; /* Menu object */

void main()
{
gApplication = new(CAltDemoApp); /* Creat appln object*/
gApplication->IApplication(4,4096L,2048L); /* Init it */
gApplication->Run(); /* Startup application */
gApplication->Exit(); /* Return to da "Finder" */
}

void CAltDemoApp::CreateDocument()
{
CAltDemoDoc *theDocument; /* New document obj */

theDocument = new(CAltDemoDoc); /* Create doc object */
theDocument->IDocument(this,FALSE); /* Init doc obj */
theDocument->NewFile(); /* Create (bogus) doc */
}

void CAltDemoDoc::NewFile(void)
{
CScrollPane *theScrollPane; /* Scroll pane object */
CAltDemoPane *theMainPane; /* Main pane object */

itsWindow = new(CWindow); /* Create window object */
itsWindow->IWindow(altWIND,FALSE,gDesktop,this);

/* Create scrolling object*/
theScrollPane = new(CScrollPane);
theScrollPane->IScrollView(itsWindow,this,10,10,0,0,
sizELASTIC,sizELASTIC, TRUE,TRUE,TRUE);
theScrollPane->FitToEncFrame(TRUE, TRUE);
theMainPane = new(CAltDemoPane);
itsMainPane = theMainPane;
itsGopher = theMainPane;
theMainPane->IEditPane(theScrollPane,this);
theScrollPane->InstallPanorama(theMainPane);

itsWindow->Select(); /* Select our window now!*/
}

void CAltDemoPane::IEditPane(CView *anEnclosure,
CBureaucrat *aSupervisor)
{
Rect margin; /* Margin rect area */

/* Setup text edit stuff */
CEditText::IEditText(anEnclosure,aSupervisor,1,1,0,0,
sizELASTIC,sizELASTIC,432);
FitToEnclosure(TRUE,TRUE);
SetRect(&margin,2,2,-2,-2);
ChangeSize(&margin,FALSE);
}

/*
 *
 * Source - AlternateCDEF.c
 * Author - Alexander S. Colwell,
 * Copyright (C) 1988, 1989
 * Purpose - This is a dual-arrow scroll bar control
 * definition procedure. It is similar to
 * the standard scroll bar except this has
 * two arrow indicators on both ends.
 */
```

```

#include <QuickDraw.h> /* QuickDraw defs */
#include <Color.h> /* Color defs */
#include <WindowMgr.h> /* Window Manager defs */
#include <ControlMgr.h> /* Control Manager defs */
#include <ToolBoxUtil.h> /* Tool Box Utilities */
#include <ColorToolBox.h> /* Color Tool Box defs */

typedef struct { /* Dual Control */
    short hasColor; /* Has color monitor flag */
    short altButton; /* Using alternate button */
    Rect thumb; /* Thumb rectangle area */
    Rect cThumb; /* Current thumb rect area */
    PolyHandle up; /* Up button handle */
    PolyHandle down; /* Down button handle */
    short bLen; /* Button length */
    short hLen; /* Half button length */
    short sLen; /* Scroll bar length */
    double tFactor; /* Thumb factor */
    short hFactor; /* Horizontal factor */
    short vFactor; /* Vertical factor */
    ControlHandle cHdl; /* Ctrl hdl to reference */
    long blackPat[2]; /* Black pattern */
    long ltGrayPat[2]; /* Light gray pattern */
} DS;

typedef DS *DSPTR;
typedef DSPTR *DSHDL;

typedef struct { /* Thumb Info */
    Rect limitRect; /* Limit rect scrolling */
    Rect slopRect; /* Slop rect scrolling */
    short axis; /* Drag axis control */
} THUMB;

typedef THUMB *THUMBPTR;

/* Misc definitions */
#define NIL (0L) /* NIL pointer */
#define abs(a) (a<0?-a:a) /* Absolute macro func */
#define min(a,b) (a<b?a:b) /* Minumim macro function */
#define max(a,b) (a<b?b:a) /* Maximum macro function */

/* Trap definitions */
#define SysEnvironsTrap 0xa090 /* System Enviroment */
#define UnknownTrap 0xa89f /* Unknown instruction */

/* System 7.0 CDEF msgs */
#define calcCntlRgn 10 /* Calculate cntl region */
#define calcThumbRgn 11 /* Calculate "thumb" rgn */
/* Control's macro procs */

#define DSFrameHUpBtn() \
    DSDrawHBtn(dPtr,dPtr->up,FALSE)
#define DSFrameHDownBtn() \
    DSDrawHBtn(dPtr,dPtr->down,FALSE)
#define DSFrameLUpBtn() \
    DSDrawLBtn(dPtr,dPtr->up,FALSE)
#define DSFrameLDownBtn() \
    DSDrawLBtn(dPtr,dPtr->down,FALSE)
#define DSFillHUpBtn() \
    DSDrawHBtn(dPtr,dPtr->up,TRUE)
#define DSFillHDownBtn() \
    DSDrawHBtn(dPtr,dPtr->down,TRUE)
#define DSFillLUpBtn() \
    DSDrawLBtn(dPtr,dPtr->up,TRUE)
#define DSFillLDownBtn() \
    DSDrawLBtn(dPtr,dPtr->down,TRUE)

#define DSFrameBody(a) \
    DSDrawObject(dPtr->cHdl,DSGetForeground, \
    DSSetForeground,DSFrameRect, \
    cFrameColor,a)
#define DSClearBody(a) \
    DSDrawObject(dPtr->cHdl,DSGetBackground, \
    DSSetBackground,DSEraseRect, \
    cBodyColor,a)
#define DSFillBody(a) \
    DSDrawObject(dPtr->cHdl,DSGetBackground, \
    DSSetBackground,DSFillRect,cBodyColor,a)

```

```

DSDrawObject(dPtr->cHdl,DSGetBackground, \
DSSetBackground,DSFillRect,cBodyColor,a, \
dPtr->ltGrayPat)
#define DSFrameThumb(a) \
    DSDrawObject(dPtr->cHdl,DSGetForeground, \
    DSSetForeground,DSFrameRect, \
    cFrameColor,a)
#define DSFillThumb(a) \
    DSDrawObject(dPtr->cHdl,DSGetBackground, \
    DSSetBackground,DSEraseRect, \
    cThumbColor,a)
#define DSFrameArrow(a) \
    DSDrawObject(dPtr->cHdl,DSGetForeground, \
    DSSetForeground,DSFramePoly, \
    cFrameColor,a)
#define DSClearArrow(a) \
    DSDrawObject(dPtr->cHdl,DSGetBackground, \
    DSSetBackground,DSErasePoly, \
    cBodyColor,a)
#define DSFillArrow(a) \
    DSDrawObject(dPtr->cHdl,DSGetForeground, \
    DSSetForeground,DSFillPoly, \
    cFrameColor,a,dPtr->blackPat)
/* Define forward refs */
void DSInitControl(),DSDrawThumb(),DSPositionThumb(), \
DSSetupThumb(),DSDrawHBtn(),DSDrawLBtn(), \
DSGetBackground(),DSGetForeground(), \
DSSetBackground(),DSSetForeground(), \
DSFrameButtons(),DSEraseRect(),DSFrameRect(), \
DSFillRect(),DSErasePoly(),DSFramePoly(), \
DSFillPoly(),DSDrawObject();
long DSTestControl();

pascal long main(varCode,theControl,message,param)
short varCode; /* Variable control code */
ControlHandle theControl; /* Control handle */
short message; /* Control message */
long param; /* Control parameter */
{
    ControlPtr cPtr; /* Working control ptr */
    DSHDL dHdl; /* Working dual ctrl hdl */
    DSPTR dPtr = NIL; /* Working dual ctrl ptr */
    Rect wRect; /* Working scroll area */
    SysEnvRec sysEnv; /* Working system envt */
    long status = 0; /* Working status flag */

    HLock(theControl); /* Lock down the ctrl hdl */
    cPtr = *theControl; /* Set control pointer */
    if (message != initCntl) { /* Check if already init */
        if (cPtr->cntlData) { /* Check if valid handle */
            HLock(cPtr->cntlData); /* Lock it down for work */
            dPtr = *((DSHDL)cPtr->cntlData); /* Set dual ptr */
        }
    }
    switch(message) { /* Process specified msg */
        case drawCntl: /* Draw the control */
            if (dPtr) { /* Check if it's valid */
                if (cPtr->cntlVis) { /* Check if it's visible */
                    PenNormal(); /* Set to normal pen */
                    switch(LoWord(param)) { /* Switch on control part */
                        case 0: /* Re-draw scroll bar */
                            DSInitControl(dPtr,cPtr); /* Re-init it */
                            DSFrameBody(&cPtr->cntlRect); /* Init scroll */
                            DSFrameButtons(dPtr); /* Frame the buttons */
                            DSFrameHUpBtn();
                            DSFrameHDownBtn();
                            DSFrameLUpBtn();
                            DSFrameLDownBtn();
                            DSDrawThumb(dPtr,theControl); break;
                        case 129: /* Value/Min/Max changed */
                            DSDrawThumb(dPtr,theControl); break;
                        case 0xff: /* Control to be inactive */
                            DSSetScrollRgn(dPtr,&wRect);
                            DSClearBody(&wRect); break;

```

```

case inPageUp: /* Hilite scroll bar */
case inPageDown:
    DSDrawThumb(dPtr,theControl); break;
case inUpButton: /* Hilite high up button */
    if (dPtr->altButton) { /* Check if alt button */
        if (cPtr->ctrlHilite == 0 ||
            cPtr->ctrlMin == cPtr->ctrlMax)
            DSFrameLUpBtn();
        else
            DSFillUpBtn();
    }
    else {
        if (cPtr->ctrlHilite == 0 ||
            cPtr->ctrlMin == cPtr->ctrlMax)
            DSFrameHUpBtn();
        else
            DSFillHUpBtn();
    }
    break;
case inDownButton: /* Hilite high down button */
    if (dPtr->altButton) { /* Check if alt button */
        if (cPtr->ctrlHilite == 0 ||
            cPtr->ctrlMin == cPtr->ctrlMax)
            DSFrameLDownBtn();
        else
            DSFillLDownBtn();
    }
    else {
        if (cPtr->ctrlHilite == 0 ||
            cPtr->ctrlMin == cPtr->ctrlMax)
            DSFrameHDownBtn();
        else
            DSFillHDownBtn();
    }
}
}
break;
case testCntl: /* Test the control */
    if (dPtr) /* Check if ptr valid */
        if (cPtr->ctrlHilite != -1) /* Check if active */
            if (cPtr->ctrlMin != cPtr->ctrlMax)
                if (PtInRect(param,&cPtr->ctrlRect))
                    status = DSTestControl(dPtr,param); /* Find it */
    break;
case calcCRgns: /* Calculate ctrl region */
    if (dPtr) { /* Check if pointer valid */
        if (!(param & 0x80000000L)) /* Check if whole ctrl */
            RectRgn((Handle)(param & 0x7fffffffL),
                &cPtr->ctrlRect);
        else { /* Want thumb's region */
            /* Setup thumb's area */
            RectRgn((Handle)(param & 0x7fffffffL),
                &dPtr->cThumb);
            /* Setup drag pattern */
            BlockMove(dPtr->ltGrayPat,DragPattern,8L);
        }
    }
    break;
case calcCntlRgn: /* Calculate cntl region */
    if (dPtr) /* Check if ptr valid */
        RectRgn((Handle)(param),&cPtr->ctrlRect);
    break;
case calcThumbRgn: /* Calculate "thumb" rgn */
    if (dPtr) { /* Check if pointer valid */
        /* Setup thumb's region */
        RectRgn((Handle)(param),&dPtr->cThumb);
        /* Setup drag pattern */
        BlockMove(dPtr->ltGrayPat,DragPattern,8L);
    }
    break;
case initCntl: /* Initialized control */
    /* Allocate data handle */
    if (dHdl = (DSHDL)NewHandle(sizeof(DS))) {

```

```

        HLock(dHdl); /* Lock it down for init */
        dPtr = *dHdl; /* Set dual control ptr */
        dPtr->cHdl = theControl; /* Save it for reference */
        dPtr->up = dPtr->down = NIL; /* Init poly handles */
        dPtr->blackPat[0] = 0xffffffffL; /* Init patterns */
        dPtr->blackPat[1] = 0xffffffffL;
        dPtr->ltGrayPat[0] = 0x88228822L;
        dPtr->ltGrayPat[1] = 0x88228822L;
        dPtr->hasColor = FALSE; /* Set to black & white */
        dPtr->altButton = FALSE; /* Not using alt button */
        /* Check SysEnviroms OK */
        if ((long)NGetTrapAddress(SysEnviromsTrap,OSTrap) !=
            (long)NGetTrapAddress(UnknownTrap,ToolTrap)) {
            SysEnviroms(1,&sysEnv); /* Get system environment */
            dPtr->hasColor = sysEnv.hasColorQD; /* Save color */
        }
        DSInitControl(dPtr,cPtr); /* Init control's stuff */
        cPtr->ctrlAction = (ProcPtr)(-1L);
        cPtr->ctrlData = (Handle)dHdl; /* Save data hdl */
    }
    break;
case dispCntl: /* Dispose the control */
    if (dPtr) { /* Check if pointer valid */
        if (dPtr->up) /* Kill the polys */
            KillPoly(dPtr->up);
        if (dPtr->down)
            KillPoly(dPtr->down);
        HUnlock(cPtr->ctrlData); /* Unlock it now */
        DisposHandle(cPtr->ctrlData); /* Return it */
    }
    cPtr->ctrlData = NIL; /* Clear it */
    break;
case posCntl: /* Re-position control */
    if (dPtr) /* Check if pointer valid */
        DSPositionThumb(dPtr,theControl,param);
    break;
case thumbCntl: /* Calculate for dragging */
    if (dPtr) /* Check if pointer valid */
        DSSetupThumb(dPtr,param); /* Setup for dragging */
    break;
case dragCntl: /* Drag the control */
    status = 0; /* Only drag thumb! */
    break;
case autoTrack: /* Execute ctrl's action */
    break;
}
if (cPtr->ctrlData) /* Check if valid pointer */
    HUnlock(cPtr->ctrlData); /* Unlock for memory mgr */
HUnlock(theControl); /* Unlock control handle */
return(status); /* Return status code */
}

void DSInitControl(dPtr,cPtr)
DSPTR dPtr; /* Dual control pointer */
ControlPtr cPtr; /* Control pointer */
{
    PolyHandle pHdl; /* Working poly handle */
    short bLen; /* Working button length */
    short hBLen; /* Working 1/2 button len */
    short hLen; /* Working horizontal len */
    short vLen; /* Working vertical len */
    short tmp1,tmp2; /* Working tmp variables */
    Rect cRect; /* Working ctrl rect area */

    cRect = cPtr->ctrlRect; /* Set scroll bar area */
    InsetRect(&cRect,1,1); /* Shrink it by one-pixel */
    /* Set horz & vert lens */
    hLen = abs(cRect.right - cRect.left) + 1;
    vLen = abs(cRect.bottom - cRect.top) + 1;

    if (hLen > vLen) { /* Check if horz longer */
        dPtr->bLen = vLen; /* Set button length */
        dPtr->sLen = hLen + 1; /* Set scroll bar length */
        dPtr->hFactor = 1; /* Set horizontal factor */

```

```

dPtr->vFactor = 0; /* Set vertical factor */
}
else { /* Nope, must be vertical */
dPtr->bLen = hLen; /* Set button length */
dPtr->sLen = vLen + 1; /* Set scroll bar length */
dPtr->hFactor = 0; /* Set horizontal factor */
dPtr->vFactor = 1; /* Set vertical factor */
}
bLen = dPtr->bLen; /* Set button length */
hBLen = dPtr->hBLen = bLen / 2; /* Set 1/2 button len */
dPtr->thumb.top = dPtr->thumb.left = 0; /* Init pt */
if (dPtr->hFactor) { /* Check if horz position */
dPtr->thumb.bottom = bLen - 1;
dPtr->thumb.right = hBLen + hBLen / 2;
}
else { /* Nope, vert position */
dPtr->thumb.bottom = hBLen + hBLen / 2;
dPtr->thumb.right = bLen - 1;
}
tmp1 = hBLen - 1; /* Set tmp calculations */
tmp2 = hBLen - 2;
/* Create up poly handle */
if (pHdl = OpenPoly()) { /* Check if got poly hdl */
if (dPtr->hFactor) { /* Check horz position */
MoveTo(0,tmp1);
LineTo(tmp2,tmp2 * 2 + 1);
LineTo(tmp2,1);
LineTo(0,tmp1);
}
else { /* Nope, vert position */
MoveTo(tmp1,0);
LineTo(1,tmp2);
LineTo(tmp2 * 2 + 1,tmp2);
LineTo(tmp1,0);
}
ClosePoly(); /* Close the poly handle */
}
if (dPtr->up) /* Check if old poly hdl */
KillPoly(dPtr->up); /* Release this poly hdl */
dPtr->up = pHdl; /* Set up poly handle */
/* Create down poly hdl */
if (pHdl = OpenPoly()) { /* Check if got poly hdl */
if (dPtr->hFactor) { /* Check horz position */
MoveTo(tmp2,tmp1);
LineTo(0,tmp2 * 2 + 1);
LineTo(0,1);
LineTo(tmp2,tmp1);
OffsetPoly(pHdl,hBLen,0);
}
else { /* Nope, vert position */
MoveTo(1,0);
LineTo(tmp2 * 2 + 1,0);
LineTo(tmp1,tmp1);
LineTo(1,0);
OffsetPoly(pHdl,0,hBLen);
}
ClosePoly(); /* Close the poly handle */
}
if (dPtr->down) /* Check if old poly hdl */
KillPoly(dPtr->down); /* Release this poly hdl */
dPtr->down = pHdl; /* Set down poly handle */
}

void DSCalcThumb(dPtr,cPtr)
DSPTR dPtr; /* Dual control pointer */
ControlPtr cPtr; /* Control pointer */
{
double tmp1,tmp2; /* Working tmp registers */
short offset; /* Working pixel offset */
Rect wRect; /* Working scroll area */

/* Make sure this is min */
cPtr->ctrlMin = min(cPtr->ctrlMin, cPtr->ctrlMax);

```

```

/* Make sure this is max */
cPtr->ctrlMax = max(cPtr->ctrlMin, cPtr->ctrlMax);

/* Make it min<value<max */
cPtr->ctrlValue = min(cPtr->ctrlMax,
max(cPtr->ctrlMin, cPtr->ctrlValue));

DSSetScrollRgn(dPtr,&wRect); /* Set scroll region */
/* Init thumb factor calc */
tmp1 = max(abs(wRect.bottom - wRect.top) * dPtr->vFactor,
abs(wRect.right - wRect.left) * dPtr->hFactor)
- dPtr->bLen + dPtr->hBLen - 2;
tmp2 = abs(cPtr->ctrlMax - cPtr->ctrlMin);

dPtr->tFactor = tmp1 / tmp2; /* Set relative factor */
tmp1 = cPtr->ctrlValue - /* Set thumb pixel offset */
cPtr->ctrlMin;
tmp2 = tmp1 * dPtr->tFactor;
offset = tmp2;
dPtr->cThumb = dPtr->thumb; /* Set thumb rect area */
OffsetRect(&dPtr->cThumb, wRect.left + offset * dPtr->
hFactor, wRect.top + offset * dPtr->vFactor);
}

void DSDrawThumb(dPtr,cHdl)
DSPTR dPtr; /* Dual control pointer */
ControlHandle cHdl; /* Control handle */
{
Rect tRect; /* Working thumb area */
ControlPtr cPtr = *cHdl; /* Working control ptr */

DSSetScrollRgn(dPtr,&tRect); /* Set scroll region */
if (cPtr->ctrlHilite != 255 && /* Check if show it */
cPtr->ctrlMax != cPtr->ctrlMin) {
DSCalcThumb(dPtr,cPtr); /* Re-calculate thumb */
ForeColor((long)(blackColor)); /* Make it's b & w */
BackColor((long)(whiteColor));
DSFillBody(&tRect); /* Fill in scroll bar */
tRect = dPtr->cThumb; /* Set thumb rect area */
DSFrameThumb(&tRect); /* Frame thumb */
InsetRect(&tRect,1,1); /* Inset by one pixel */
DSFillThumb(&tRect); /* Clear inside of thumb */
}
else /* Nope, don't want it */
DSClearBody(&tRect); /* Clr scroll region */
}

void DSPositionThumb(dPtr,cHdl,pt)
DSPTR dPtr; /* Dual control pointer */
ControlHandle cHdl; /* Control handle */
Point pt; /* Pt position of thumb */
{
double offset; /* Working value offset */

if (dPtr->hFactor) /* Check if horizontal */
offset = pt.h; /* Use horizontal offset */
else /* Nope, it's vertical */
offset = pt.v; /* Use vertical offset */
offset = offset / dPtr->tFactor; /* Reset ctrl value */
(*cHdl)->ctrlValue += offset;
DSDrawThumb(dPtr,cHdl); /* Draw the thumb, again! */
}

void DSSetupThumb(dPtr,thumbPtr)
DSPTR dPtr; /* Dual control pointer */
THUMBPTR thumbPtr; /* Thumb's info pointer */
{
Point msePt; /* Working mouse point */

msePt.h = thumbPtr->limitRect.left; /* Save mse down */
msePt.v = thumbPtr->limitRect.top;
DSSetScrollRgn(dPtr,&thumbPtr->limitRect); /* Set limit */
if (dPtr->hFactor) { /* Check if horz scroll */
/* Adjust it for mse */

```



```

thumbPtr->limitRect.left += msePt.h - dPtr->cThumb.left;
thumbPtr->limitRect.right -= dPtr->cThumb.right - msePt.h - 1;
/* Set slop rect area */
thumbPtr->slopRect.top = thumbPtr->limitRect.top - 16;
thumbPtr->slopRect.bottom = thumbPtr->limitRect.bottom +
16;
thumbPtr->slopRect.left = -32000;
thumbPtr->slopRect.right = 32000;
thumbPtr->axis = hAxisOnly; /* Set axis dragging dir*/
}
else { /* Nope, vert scroll bar*/
/* Adjust it for mse */
thumbPtr->limitRect.top += msePt.v - dPtr->cThumb.top;
thumbPtr->limitRect.bottom -= dPtr->cThumb.bottom - msePt.v
- 1;
/* Set slop rect area */
thumbPtr->slopRect.left = thumbPtr->limitRect.left - 16;
thumbPtr->slopRect.right = thumbPtr->limitRect.right + 16;
thumbPtr->slopRect.top = -32000;
thumbPtr->slopRect.bottom = 32000;
thumbPtr->axis = vAxisOnly; /* Set axis dragging dir*/
}
}

```

```
void DSDrawHBtn(dPtr,btn,fill)
```

```

DSPTR dPtr; /* Dual control pointer */
PolyHandle btn; /* Poly button handle */
short fill; /* Fill flag indicator */

{
/* Offset where button */
OffsetPoly(btn,(*dPtr->cHdl)->ctrlRect.left + 1,
(*dPtr->cHdl)->ctrlRect.top + 1);
if (fill) /* Check if filling arrow*/
DSFillArrow(btn); /* Fill in arrow button */
else /* Nope, not clearing it */
DSClearArrow(btn); /* Clear the arrow button*/
DSFrameArrow(btn); /* Frame the arrow button*/
/* Restore the button */
OffsetPoly(btn,-(*dPtr->cHdl)->ctrlRect.left - 1,
-(*dPtr->cHdl)->ctrlRect.top - 1);
}

```

```
void DSDrawLBtn(dPtr,btn,fill)
```

```

DSPTR dPtr; /* Dual control pointer */
PolyHandle btn; /* Poly button handle */
short fill; /* Fill flag indicator */

{
short tmp1,tmp2; /* Working tmp variables */
short offset; /* Working offset */

offset = dPtr->sLen - dPtr->bLen - 1; /* Set offset */
/* Offset where button */
OffsetPoly(btn,tmp1 = (*dPtr->cHdl)->ctrlRect.left +
dPtr->hFactor * offset + 1,tmp2 = (*dPtr->cHdl)-
>ctrlRect.top + dPtr->vFactor * offset + 1);
if (fill) /* Check if filling arrow */
DSFillArrow(btn); /* Fill in arrow button */
else /* Nope, not clearing it */
DSClearArrow(btn); /* Clear the arrow button */
DSFrameArrow(btn); /* Frame the arrow button */
OffsetPoly(btn,-tmp1,-tmp2); /* Restore the button */
}

```

```
void DSFrameButtons(dPtr)
```

```

DSPTR dPtr; /* Dual control pointer */

{
Rect bRect; /* Working box rect area */
short offset; /* Working low button */

bRect.left = bRect.top = 0; /* Setup rect area */
bRect.right = bRect.bottom = dPtr->bLen + 1;
/* Offset high button box*/

```

```

OffsetRect(&bRect,(*dPtr->cHdl)->ctrlRect.left,
(*dPtr->cHdl)->ctrlRect.top);
DSClearBody(&bRect); /* Clear the body part */
DSFrameBody(&bRect); /* Frame high button box */
/* Set low button offset */
offset = dPtr->sLen - dPtr->bLen - 1;
OffsetRect(&bRect,dPtr->hFactor * offset,
dPtr->vFactor * offset);
DSClearBody(&bRect); /* Clear the body part */
DSFrameBody(&bRect); /* Frame low button box */
}

```

```
long DSTestControl(dPtr,pt)
```

```

DSPTR dPtr; /* Dual control pointer */
Point pt; /* Point position */

{
Rect cRect; /* Working control rect */
Rect tRect; /* Working temp rect area*/
Rect bRect; /* Working button rect */
short offset; /* Working offset */
short point; /* Working pt in region */
long status = 0; /* Working status flag */

```

```

cRect = (*dPtr->cHdl)->ctrlRect; /* Set ctrl rect */
bRect = dPtr->thumb; /* Set button rect area */
if (dPtr->hFactor) /* Check if horz button */
bRect.right = dPtr->hBLen; /* Adjust for horz button*/
else
bRect.bottom = dPtr->hBLen; /* Adjust for vert btn*/
tRect = bRect; /* Set the temp rect area*/
OffsetRect(&tRect,cRect.left,cRect.top);
if (PtInRect(pt,&tRect)) { /* Check if high up btn*/
status = inUpButton; /* Set high up button */
dPtr->altButton = FALSE; /* Not using alt button */
}
if (!status) { /* Check if prev part */
/* Adjust for down button*/
OffsetRect(&tRect,dPtr->hFactor * dPtr->hBLen,
dPtr->vFactor * dPtr->hBLen);
if (PtInRect(pt,&tRect)) { /* Check if high down */
status = inDownButton; /* Set high down button */
dPtr->altButton = FALSE; /* Not using alt button */
}
}
if (!status) { /* Check if prev part */
tRect = bRect; /* Set the temp rect area*/
offset = dPtr->sLen - dPtr->bLen;
/* Offset where low up */
OffsetRect(&tRect,cRect.left + dPtr->hFactor*offset,
cRect.top + dPtr->vFactor*offset);
if (PtInRect(pt,&tRect)) { /* Check if low up button*/
status = inUpButton; /* Set low up button */
dPtr->altButton = TRUE; /* Using alt button */
}
}
if (!status) { /* Check if prev part */
/* Offset where low down */
OffsetRect(&tRect,dPtr->hFactor * dPtr->hBLen,
dPtr->vFactor * dPtr->hBLen);
if (PtInRect(pt,&tRect)) { /* Check if the low down */
status = inDownButton; /* Set low down button */
dPtr->altButton = TRUE; /* Using alt button */
}
}
if (!status) { /* Check if prev part */
/* Check if thumb */
status = inThumb; /* Set thumb part */
}
if (!status) { /* Check if prev part */
/* Set relative point */
point = pt.h * dPtr->hFactor + pt.v * dPtr->vFactor;
/* Set relative offset */
offset = dPtr->cThumb.left * dPtr->hFactor +

```

```

    dPtr->cThumb.top * dPtr->vFactor;
    if (point > offset) /* Check if in down page */
        status = inPageDown; /* Set down page region */
    else
        status = inPageUp; /* Set up page region */
    return(status); /* Return status part */
}

```

```

DSSetScrollRgn(dPtr,scrollRgn)
DSPTR dPtr; /* Dual control pointer */
Rect *scrollRgn; /* Scrolling region area */
{
    *scrollRgn = (*dPtr->cHdl)->controlRect; /* Set area */
    InsetRect(scrollRgn,1,1); /* Shrink it a bit */
    if (dPtr->hFactor) /* Check if horizontal bar */
        scrollRgn->left += dPtr->bLen; /* Adjust rect area */
        scrollRgn->right -= dPtr->bLen;
    }
    else { /* It's a vert scroll bar */
        scrollRgn->top += dPtr->bLen; /* Adjust rect area */
        scrollRgn->bottom -= dPtr->bLen;
    }
}

```

```

void DSGetBackGround(color)
    RGBColor *color; { GetBackColor(color); }
void DSGetForeGround(color)
    RGBColor *color; { GetForeColor(color); }
void DSSetBackGround(color)
    RGBColor *color; { RGBBackColor(color); }
void DSSetForeGround(color)
    RGBColor *color; { RGBForeColor(color); }

```

```

void DSEraseRect(rect)
    Rect *rect; { EraseRect(rect); }
void DSFrameRect(rect)

```

```

    Rect *rect; { FrameRect(rect); }
void DSFillRect(rect,pattern)
    Rect *rect; Pattern *pattern; { FillRect(rect,pattern); }
void DSErasePoly(poly)
    PolyHandle poly; { ErasePoly(poly); }
void DSFramePoly(poly)
    PolyHandle poly; { FramePoly(poly); }
void DSFillPoly(poly,pattern)
    PolyHandle poly; Pattern *pattern;
    { FillPoly(poly,pattern); }

```

```

void DSDrawObject(cHdl,getGround,setGround,
    object,color,arg1,arg2)
    ControlHandle cHdl; /* Control handle */
    ProcPtr getGround; /* Get back/fore grd proc*/
    ProcPtr setGround; /* Set back/fore grd proc*/
    ProcPtr object; /* Draw object proc */
    short color; /* Color index */
    long arg1,arg2; /* Argument's parameters */
{
    AuxCtlHndl acHdl = NIL; /* Working aux color hdl */
    RGBColor oldColor; /* Working old color */
    RGBColor newColor; /* Working new color */
    /* Check if has color */
    if (((DSHDL)((*cHdl)->controlData))->hasColor) {
        GetAuxCtl(cHdl,&acHdl); /* Get control color info*/
        if (acHdl) { /* Check if really got it*/
            (*getGround)(&oldColor); /* Get back-ground color */
            /* Get object's color */
            newColor = ((*acHdl)->acCTable)->ctTable[color].rgb;
            (*setGround)(&newColor); /* Set control obj color */
        }
    }
    (*object)(arg1,arg2); /* Draw the object */
    if (acHdl) /* Check if have aux hdl */
        (*setGround)(&oldColor); /* Restore back-grd color*/
}

```



Jörg's Folder

Apple's C++



Jörg Langowski
MacTutor Editorial Staff
Volume 5, Number 11

"C++"

OOPS has taken its literal meaning with the brief C++ introduction that I wrote 2 months ago... now that I have a compiler to test the examples that I thought up. Of course, I did it all wrong. Seasoned C++ experts might have smiled when they read my 'matrix class' definitions, others (like myself) might not have noticed the goofups at all. Anyway, this month you'll find much more detailed examples of the same kind, and I can assure you they work, because they've been tested. [Before you cancel your subscription in panic, let me emphasize that the overview of C++ in the same article is still basically correct].

Someone on the networks came up with this quote: "C++ is the Forth of the 90's - they don't call it 'uh-oh programming' for nothing". Well, I can confirm on the uh-oh part, and as far as the Forth is concerned, at least I'm learning this language as much by trial and error as I did with Forth. So let's follow the learning curve together; maybe that is not such a bad approach for a tutorial.

Our example is an MPW tool that does some array computations using a **matrix** and **vector** class that have a number of methods defined. We'll go through the example step by step, and learn some important features of data structure definition in C++. The program is not Mac-specific; it uses only the standard C++ libraries, so you can check out the examples on a UNIX machine with C++ even if you don't have access to Apple's C++ (and if you do, chances are that you might not need this tutorial at all).

Simple I/O in C++

The first few lines of our example - after the #includes - define a simple error message function which writes a string to standard error and exits the program. The definition looks very much like C except for the line

```
cerr << p << endl;
```

which is a C++ I/O statement that writes the string **p** to standard error. **cerr** is an object of class **ostream** which is basically an output file; **cerr** is predefined to be the standard error output for a UNIX system or for MPW. Likewise, **cin** and **cout** are predefined standard input and output (console). **<<** is the output operator, and to write an object **x** to standard output you simply write

```
cout << x;
```

Whether the compiler will accept this statement or not depends whether the operator **<<** has been defined for the type **x**. Standard definitions for most simple types are contained in the header file **Stream.h**. The operator **>>** is the input operator:

```
cin >> i;
```

would read the int **i** from standard input. **endl** is used to flush the output buffer (otherwise the actual output might appear on the screen at a later time), and add an end-of-line.

The Vector Class

In the following, we define a class **vector**, a resizable one-dimensional array of **float** with optional range checking. Its private variables are **float* v**, which will be used as a pointer to the data, and **int sz**, the number of elements in the vector.

We then declare the class **matrix** (defined further down) as a **friend** of **vector**. Why this? Later, we will define an operator which multiplies a matrix with a vector as a method of the class **matrix**. This operator needs access to the data contained in the vector. Since the normal way of accessing involves bounds checking and is therefore slow, we want to be able to access the 'private part' **v** of the vector directly. Friends of a class may do this.

The publicly accessible parts of the class are all the methods defined for our vector:

- the constructors, **vector(int)** and **vector(vector&)**; the first one creates a new vector given its size as an argument, the second one duplicates an existing vector by creating a new one of the same size and copying all the elements;
- the destructor, **~vector()**, which will free the space allocated to the object's data when the object is no longer needed;
- **size()**, an inline function that returns the length of the vector;
- **set_size(int)**, which changes the size of an existing vector;
- the indexing operator **[]**, for accessing an element of the vector. We have to note here that the operator returns a reference to a **float**, not a **float** value, so that an indexed element can be used on both sides of an assignment statement. **[]** will check the index against the bounds of the vector;
- **elem(int i)**, an inline function which returns the value of the *i*-th element of the vector without range checking, for fast access;
- the operators **=**, **+**, and **-**, which are used for assignment and elementwise addition or subtraction of vectors. They will check whether the vectors on both sides of the operator sign are of equal size, perform the operation, and return the reference to a newly created vector which contains the result;
- the operator *****, which computes the scalar product of two vectors, and returns a **float** value;
- **print()**, which prints out the contents of the vector to **cout**.

The class definition constitutes the interface to the object's structure and its methods; the actual implementation follows.

I have put informative messages into the constructor and destructor methods so that one can see how new objects are created and released. This is important for debugging; if less objects are released than created, something might have gone wrong, and at least one has wasted memory space.

vector(size) creates an array of **floats** on the heap that is **size** elements long. **vector(a)**, where **a** is another vector, will create a

new vector which is an exact copy of `a`. These constructors are called when new vectors are defined:

```
vector x(5); // calls vector(int)
```

```
vector y = x; // calls vector(vector&)
```

In the `for` statement you see another characteristic feature of C++: declarations can be made anywhere, so the loop index `i` may be declared as a register variable just for the scope of the loop.

The destructor `~vector()` is called when an instance of `vector` goes 'out of scope', and deletes the space allocated for the array `v` before the object itself is deleted. Defining a destructor is usually done when you have allocated some of your own memory on construction, just as we did.

The operator `[]` will first see if the index is within the limits of the array and abort with an error message if not (this could be done in a more sophisticated way); then it indexes the private array `v` and returns a reference to its `i`-th element. Note that the operator `[]`, when used with `v`, will be the usual array indexing function; when used on a vector, it uses range checking, since its original definition has been overloaded. If we define a function or an operator with the same name as an existing one, old definitions will automatically be overloaded in C++ 2.0; the overload statement of the previous version described in Stroustrup's book is not necessary anymore.

The assignment operator will copy all elements of the argument vector into its parent vector, making sure that both are of the same size. It returns a reference to the argument vector, so that multiple assignment statements are possible:

```
a = b = c; // copies c into b and a
```

`+` and `-` create a new vector of the same size as of its arguments and copies the sum of the two vectors into it. The new vector is allocated on the heap, using the `new` operator:

```
vector& c = *new vector(sz);
```

that is, we define a reference `c` to a vector, to which we assign the pointer to the new vector. One could think that one could simply create a new instance by writing

```
vector c(sz);
```

however, this will only create a local instance of `vector`, which will be deleted as soon as the function is left. The compiler, by the way, complains if you use a local object as the return value (just as a warning, it will still compile...). Rightly so, since local objects are allocated in a local stack frame, and disappear as soon as that stack frame is UNLKed and the JSR called.

Therefore, we must create a new vector in the heap which will hold the result of the addition or subtraction. This will later require some housekeeping on our part, because a heap object is not automatically deleted when its scope is left. We must take care of that ourselves (see below).

The definition of the scalar product, `*`, and the `print()` function should now be self-explanatory. I have not given a definition for the resizing function, `set_size(int)`; try whether you can come up with a definition for it.

The Matrix Class

Contrary to what I wrote in the September column, we won't define `matrix` as a class derived from `vector`. Also, we won't define `matrix` as an array of vectors; I had tried this approach, which is very elegant when it works, but screwed up so com-

pletely that I haven't come up with a solution as the deadline is approaching. Therefore, we'll use a simpler definition, which has the advantage of being faster than what I thought up before.

Thus: class `matrix` has the private variables `mv`, a pointer to an array of floats, and the two integers `rows` and `cols`, which indicate the number of rows and columns in the array. We will do our index calculations explicitly in the methods of `matrix` and use one-dimensional indexing for actually accessing the elements of `mv` within the method. To the outside, this is not visible and we'll always use two-dimensional indexing with range checking.

The methods defined are:

- constructors and a destructor, analogous to those of class `vector`;
- two functions that return the number of rows and columns of the matrix (note that `rowsize` should not be read as 'size of one row', but 'size in units of rows');
- `set_size`, unimplemented as for `vector`, left for you as an exercise;
- the two-dimensional indexing operator `()`, which takes two ints `i,j` as arguments and returns a reference to the `(i,j)`-th element. Thus, we won't use square brackets for indexing: instead of `a[i][j]` like in C we'll write `a(i,j)` like in Fortran. This is an overload of the function call operator `()` which was not possible in version 1 of C++, but is legal in C++ 2.0. Any operator may be redefined in C++ 2.0, including the comma.
- the quick indexing function `elem(i,j)`, which returns the value of the `(i,j)`-th element without bounds checking;
- the assignment, addition and subtraction operators, which work analogously to those for vectors;
- two multiplication operators, one for multiplying two matrices, one for multiplying a vector by a matrix from the left;
- the `print()` function.

The method definitions follow the class declaration, and you should now be able to go through them without much explanation. The `matrix*vector` operator accesses the data of the vector argument `a` directly (`a.v[j]`); it is for this reason that we made `matrix` a friend of `vector`. We could have achieved the same result without the friend declaration by using the inline indexing function `a.elem(j)`, but the inline expansion of function is not guaranteed in all implementations.

The Main Program

Using matrix operations in the main program has now become very simple and easy to read. For instance, the statement

```
matrix& x = a*c;
```

will compute the matrix product of `a` and `c` (given that the dimensions are correct such that the product can be computed), put the result into a new matrix allocated on the heap and make the reference `x` point to that matrix. From now on, `x` can be used whenever an object of type `matrix` is required; `x.print()`, for instance, prints the contents of `x`. The only thing that we have to be careful about is to delete all 'hidden instances' explicitly at the end of the block in which they have been created:

```
delete &x; delete &f;
```

The compiler will also issue warnings here, presumably

because x and f might not be references to heap objects, and the delete statements might crash at run time.

More complicated statements like

```
matrix& x = (a + b)*c;
```

might leave intermediate results on the heap to which there is no more access. Some sort of a garbage collector would be needed to clean up those objects; I don't know whether something is planned in that direction. For the time being, you can always break up such lines into multiple statements and assign any intermediate result explicitly to a reference variable, which can be deleted later.

If you run the example - and we include the MPW tool on the source code disk for those who have no C++ compiler available - you'll notice how instances of vectors and matrices are created and deleted, and how the product matrix A*C automatically comes out the correct size.

In the future, I am planning to implement a whole library of matrix functions in C++; maybe we'll have a follow-up on this article some time. In the next part of the tutorial, we'll deal with Mac-style programming, we'll define a simple application class with an event loop and event handlers.

Apple's C++

Although I haven't covered Macintosh-specific details this month, let me just mention the fact that the examples given here were compiled using a test version of the Apple MPW C++ compiler. This compiler is based upon the AT&T C++ Language System, which is a pre-compiler that generates C code from a C++ source. The C code can then be compiled into machine code by any C compiler. This strategy was taken so that the C++ system could be quickly implemented on any machine running C.

The actual version of AT&T C++ is 2.0, and deviates from the description in Bjarne Stroustrup's book. A number of features have been added, which I have partially covered already. As we encounter them in the course of these tutorials, we'll learn more about the changes in C++ 2.0 with respect to the book.

Feedback dept.

"Dear Jörg,

Thank you for your interesting article on compilers in August MacTutor. I feel very little attention has been paid to the quality of the code produced by Mac compilers. I have been porting some FORTRAN code using LSF 1.2 to the MacII and have been very embarrassed by IBM'ers when comparing execution speed on their 16MHz 386 machines - due purely to the much better optimization of MS-DOS FORTRAN compilers *[Having used Lahey Fortran 77 on a 286 machine, I can only agree with you - jl]*.

I was also very disappointed when Apple released MPW 3.0 to find that their C compiler produced code significantly slower than the old 2.0C compiler from Green Hills. For example the old 2.0 compiler ran your benchmarks (as written) in 3.8 (non opt) and 2.8 (opt) secs respectively. Using pointers instead of indexing in the inner loop reduced that to 2.46 secs. Happily one can use the 2.0 compiler as a tool under 3.0 for those time critical routines and mix 2.0 and 3.0 object code as long as you follow

certain rules *[Summarize those rules for us and write us about it, this is interesting - jl]*.

I suppose you have had your attention drawn to the bugs in the C code, but just in case.....

- C arrays index from 0 so you were overwriting memory when addressing index[50][50] in your C routine!

[True! What shall I say except the remarks about learning curves that I made in the beginning.....]

- C and Pascal arrays are reversed as compared to FORTRAN. i.e. A(20,30) in FORTRAN should be declared A[30][20] in C. It follows from this that for correct addressing the arrays in the C parameter list should be declared a[][50], not a[50][]. I found that if you correct this then the C non-pointer routines are significantly slowed down even more!

Andrew Cunningham, Mona Vale, NSW, Australia"

[The historical reason for the unsatisfactory optimization of most Macintosh compilers is probably that in the beginning most people were so glad to have any compiler at all that they did not pay much attention to the quality of the code as long as the stuff ran at all. Also, the toolbox calls are always the same, so it doesn't really matter whether your code is well-optimized or not as long as you mainly use toolbox calls and not doing any involved computations. Hopefully, the impact of the MacII in the scientific/engineering sector is strong enough to create some pressure on compiler writers to provide optimal tools - jl]

On the network:

Date: Fri, 8 Sep 89 10:18:24 CDT

From: "Bob Loewenstein" <rlf@oddjob.uchicago.edu>

To: Langowski%frembl51.bitnet@tank.uchicago.edu

Subject: Neon

I talked briefly with Kriya and it looks like they will put it into the public domain shortly.

Bob

[Very good! We're all looking forward to it!]

See you again next month.

Listing 1: Matrix operations in C++

```
#include <String.h>
#include <Stream.h>
#include <StdLib.h>

void error (char* p)
{
    cerr << p << endl;
    exit(1);
}

class vector
{
    float* v;
    int sz;
    friend matrix;

public:
    vector(int);
    vector(vector&);
```

```

~vector();
int size () { return sz; }
void set_size(int);
float& operator[](int);
float elem(int i) { return v[i]; }
vector& operator= (vector&); // assignment
float operator* (vector&); // scalar product
vector& operator+ (vector&); // elementwise addition
vector& operator- (vector&); // and subtraction
void print();
};

vector::vector (int size)
{
    cout << "constructing vector object" << endl;
    if (size<=0)
        { v = 0; return; }
    sz = size;
    v = new float[sz];
}

vector::vector (vector& a)
{
    cout << "copying vector object" << endl;
    sz = a.sz;
    v = new float[sz];
    for (register int i=0 ; i<sz ; i++)
        v[i] = a.v[i];
}

vector::~vector ()
{
    cout << "deleting vector object" << endl;
    delete v;
}

float&
vector::operator[] (int i)
{
    if (i<0 || i>= sz)
        error("class vector: index out of range");
    return v[i];
}

vector&
vector::operator= (vector& a)
{
    if (a.sz != sz)
        error("class vector: size mismatch in =");
    for (register int i=0 ; i<sz ; i++)
        v[i] = a.v[i];
    return a;
}

vector&
vector::operator+ (vector& a)
{
    if (sz != a.sz)
        error("class vector: size mismatch in +");
    vector& c = *new vector(sz);
    // DON'T USE vector c(sz);
    // this would be destroyed on exiting the method
    for (register int i=0 ; i<sz ; i++)
        c.v[i] = v[i]+a.v[i];
    return c;
}

vector&
vector::operator- (vector& a)
{
    if (sz != a.sz)
        error("class vector: size mismatch in -");
    vector& c = *new vector(sz);

```

```

    for (register int i=0 ; i<sz ; i++)
        c.v[i] = v[i]-a.v[i];
    return c;
}

float
vector::operator* (vector& a)
{
    if (sz != a.sz)
        error("class vector: size mismatch in *");
    float sum = 0;
    for (register int i=0 ; i<sz ; i++)
        sum = sum + v[i]*a.v[i];
    return sum;
}

void
vector::print()
{
    for (int i=0; i<sz-1; i++)
        cout << v[i] << " | ";
    cout << v[sz-1] << endl;
}

class matrix
{
    float* mv;
    int rows, cols;

public:
    matrix(int,int);
    matrix(matrix&);
    ~matrix();
    int rowsize () { return rows; }
    int colsize () { return cols; }
    void set_size(int,int);
    float& operator()(int,int);
    float elem(int i, int j) { return mv[i*cols + j]; }
    matrix& operator= (matrix&); // assignment
    matrix& operator+ (matrix&); // elementwise addition
    matrix& operator- (matrix&); // elementwise subtraction

    matrix& operator* (matrix&); // matrix product
    vector& operator* (vector&); // matrix*vector
    void print();
};

matrix::matrix (int rowsz, int colsz)
{
    cout << "constructing matrix object" << endl;
    if (rowsz <=0 || colsz <=0)
        { mv = 0; return; }
    rows = rowsz;
    cols = colsz;
    mv = new float[rows*cols];
}

matrix::matrix (matrix& a)
{
    cout << "copying matrix object" << endl;
    rows = a.rows; cols = a.cols;
    int size = rows*cols;
    mv = new float[size];
    for (register int i=0 ; i<size ; i++)
        mv[i] = a.mv[i];
}

matrix::~matrix ()
{
    cout << "deleting matrix object" << endl;
    delete mv;
}

float&

```

```

matrix::operator() (int i, int j)
{
    if (i<0 || i>= rows)
        error("class matrix: column index out of range");
    if (j<0 || j>= cols)
        error("class matrix: row index out of range");
    return mv[i*cols + j];
}

matrix&
matrix::operator= (matrix& a)
{
    if ((a.rows != rows) || (a.cols != cols))
        error("class matrix: size mismatch in assignment");
    for (register int i=0 ; i<rows*cols ; i++)
        mv[i] = a.mv[i];
    return a;
}

matrix&
matrix::operator+ (matrix& a)
{
    if ((rows != a.rows) || (cols != a.cols))
        error("class matrix: size mismatch in +");
    matrix& c = *new matrix(rows,cols);
    for (register int i=0 ; i<rows*cols ; i++)
        c.mv[i] = mv[i]+a.mv[i];
    return c;
}

matrix&
matrix::operator- (matrix& a)
{
    if ((rows != a.rows) || (cols != a.cols))
        error("class matrix: size mismatch in -");
    matrix& c = *new matrix(rows,cols);
    for (register int i=0 ; i<rows*cols ; i++)
        c.mv[i] = mv[i]-a.mv[i];
    return c;
}

matrix&
matrix::operator* (matrix& a)
{
    if (cols != a.rows)
        error("class matrix: size mismatch in *");
    matrix& c = *new matrix(rows,a.cols);
    for (register int i=0 ; i<rows; i++)
        for (register int j=0 ; j<a.cols; j++)
        {
            register float sum = 0;
            for (register int k=0 ; k<cols; k++)
                sum = sum + elem(i,k)*a.elem(k,j); // elem is faster
            c(i,j) = sum;
        }
    return c;
}

vector&
matrix::operator* (vector& a)
{
    if (cols != a.sz)
        error("class matrix: size mismatch in *vector");
    vector& c = *new vector(rows);
    for (register int i=0 ; i<rows; i++)
    {
        register float sum = 0;
        for (register int j=0 ; j<cols; j++)
            sum = sum + elem(i,j)*a.v[j];
        c[i] = sum;
    }
    return c;
}

```

```

void
matrix::print()
{
    for (int i=0; i<rows; i++)
    {
        for (int j=0; j<cols-1; j++)
            cout << elem(i,j) << " | ";
        cout << elem(i,cols-1) << endl;
    }
}

int
main(int argc, char* argv[])
{
    cout << "Start main program..." << endl;

    int i,j;

    cout << "Defining A and C" << endl;
    matrix a(3,5);
    matrix c(5,3);

    for (i=0; i<3; i++)
        for(j=0; j<5; j++)
        {
            a(i,j) = i + (1.0*j)/10;
            c(j,i) = a(i,j) - 2;
        }

    cout << "Assigning B" << endl;
    matrix b = a;

    cout << "Elements of A:" << endl;
    a.print();

    cout << "Elements of B:" << endl;
    b.print();

    cout << "Elements of C:" << endl;
    c.print();

    //cout << "Defining X" << endl;
    //matrix x(3,3);

    cout << "Multiplying A*C into X" << endl;
    matrix& x = a*c;

    cout << "Elements of A*C:" << endl;
    x.print();

    cout << "Defining D" << endl;
    vector d(3);
    for (i=0; i<3; i++)
        d[i] = i*2.75;

    cout << "Defining E" << endl;
    vector e = d;

    cout << "Multiplying C*E into F" << endl;
    vector& f = c*e;

    cout << "Elements of D, E, F:" << endl;
    d.print(); e.print(); f.print();

    a(1,4) = 432.056;
    b = a;

    a.print(); b.print();

    delete &x; delete &f;

    cout << "Exiting" << endl;
    return 0;
}

```



Intermediate Mac'ing

AppleTalk Chatter Box



Rob Hafernik
Austin, TX
Volume 5, Number 12

A Chat Program for AppleTalk

(Rob Hafernik has been programming at one job or another for 10 years. Over the years, he's been involved in a wide variety of programming projects; working with banks, hospitals, the Space Shuttle, robotics, graphics, and so on. He's finally lucked out and now actually gets paid to program the Macintosh for Technology Works in Austin, Texas. He can be reached via e-mail on BIX or MacNet as user ID "rob42")

From the first time I sat down to a Mac connected to AppleTalk, I wanted to write a program to let users on AppleTalk-connected Macs chat back and forth. The idea languished in the back of my mind for years until I recently found myself with a need to learn AppleTalk and a whole day free of other programming responsibilities. Thus Chatterbox was begun.

I had already spent a couple of hours studying *Inside AppleTalk* to get a feel for the various AppleTalk protocols and more time pouring over the various *Inside Macintosh* chapters that cover AppleTalk. From the start, I'd decided that the program would not be restricted to just two users talking back and forth. The program had to be able to let any user send a message to any subset of the rest of the users on the internet (or all of them). I also wanted to keep the program as simple and fool-proof as possible. From all of this I came to the decision that Chatterbox would use the Name Binding Protocol (NBP) to see who was available to talk to and the Datagram Delivery Protocol (DDP) to send and receive the actual messages.

AppleTalk Basics

First a few AppleTalk terms. Each device (such as your Mac or LaserWriter) on an AppleTalk network is called a *node*. Up to 32 devices can be connected together to form an AppleTalk *network* (up to 255 devices are allowed with Apple's new release of AppleTalk). Several networks may be connected together (via special nodes called *internet routers*) to form a *zone*. The sum of all of the networks connected together are referred to as an *internet*. As each device boots up, it establishes contact with its network and comes up with a unique node id. Since more than one program may be running on a given node that wants to use the network, an addressing mechanism called *sockets* allows each node to have up to 254 separate addressable entities running at any one time. The low numbered sockets are reserved by Apple for the low-level protocols (such as the Name Binding Protocol we'll discuss below), but sockets 128-254 are open for use by any Mac program. Each socket must have a *socket listener*. This is a piece of code that the DDP calls to handle incoming messages. Apple supplies a default socket listener that does the minimum - it copies incoming messages to the buffer

you supply and sets a result field.

The Datagram Delivery Protocol is the simplest data transfer protocol of them all and the one that all of the higher-level protocols are based on: it doesn't guarantee delivery and it doesn't guarantee that packets will arrive intact. What does it do? Well, it provides for the "best effort" delivery of a packet of up to 586 bytes to a given network, node and socket on the internet. What more could a chat program ask for? One note: throughout this program I have used the "alternate interface", as opposed to the "preferred interface". The "preferred interface" (described in *Inside Macintosh Volume V*) is based on the same type of parameter-block calls as the low-level file system interface. Unfortunately, since the "preferred interface" provides no equivalent of the "DDPRead" function, a socket listener and read function must be provided by the programmer. For this simple application, the "alternate interface" proved best and easiest to understand.

The Name Binding Protocol provides a way for programs that will use AppleTalk (*client programs*) to register on the network. Each client program opens a socket, then registers a name and type with the NBP. A client program can also ask NBP for the names, types, and addresses of other clients in its zone. NBP provides a wild-card match scheme that allows, for example, a client to scan the zone for all other names of a given type. This is the mechanism that Chatterbox uses to find other users to talk to.

The Chatterbox Program

Now let's look at the program. When Chatterbox first starts up, it asks the user for the name they wish to use during the session (figure 1). The name defaults to the Chooser name, but the user may override this and provide any name less than 32 characters long (the limit allowed by the NBP). After that, the program displays a window with several parts (figure 2). There is a scrollable area where the names of other Chatterbox users are displayed; an area for status messages (I like programs that let me know what they're doing, especially ones like this whose actions are mostly invisible); a TextEdit field for messages you wish to send; and a read-only TextEdit field where incoming messages

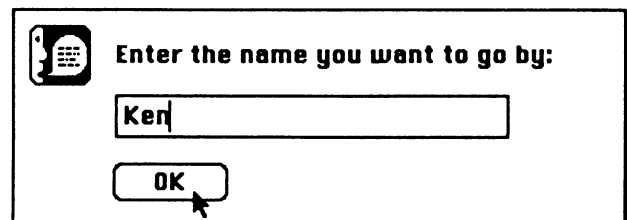


Figure 1

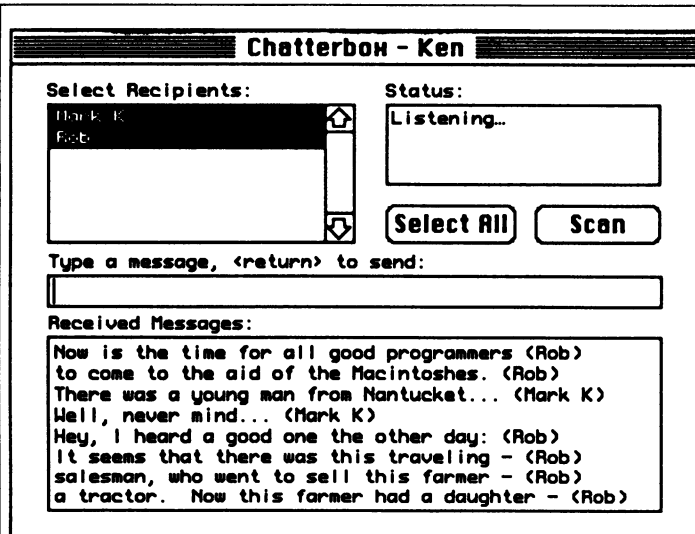


Figure 2

are displayed.

To use the program, the user selects recipients from the list. Multiple selections are allowed, both continuous selections made with a shift-click and discontinuous selections made with a command-click. Messages entered in the edit-text field are transmitted by hitting the <return> key at the end of the message. As others send messages, they're displayed in the area at the bottom. As more messages come in, the old ones scroll up and are eventually deleted as they scroll off the top.

The flow of the program is pretty simple. Once the usual Mac stuff is initialized, the program opens a socket that will be used to receive messages. To keep things easy, I used the default, Apple-supplied, socket listener. Then the RegisterName routine allocates a parameter block, fills in the user's name, type, zone, socket and a few other parameters and calls NBPRRegister to do the work. The zone string is set to "*", meaning "this zone" and the type string to "Chatterbox". By using the "*" zone name, Chatterbox is limiting communications to a single zone (the one it's in), but this simplifies the program considerably. Note that when the program quits, it must call NBPRRemove to take this name out of the list - see the UnRegisterName routine.

After registering on the internet, the next thing the program does is send out a special greeting to all other Chatterboxes. Why? Well, it's a long story. When I first wrote the program, I thought that it would periodically scan the zone for new users and add them to its display. This proved to be very cumbersome (it takes a second or two to do the lookup for one thing, a pretty big interruption if you're doing anything) and I went through several schemes before I came up with another method: as Chatterbox gets incoming messages, it looks to see if they begin with the non-breaking space (option-space from the keyboard). If so, it's a signal that the list of users in the zone has changed and it's time to call CheckForUsers (discussed below). This means that the user is only interrupted when a new user comes along (or goes away, we'll get to that later). So this special message that is sent out by a new Chatterbox coming along triggers all the other Chatterboxes to look around the zone for their fellows. Why go

to all the trouble of a CheckForUsers and not just add the new user to the list? In a word, redundancy. This may take a little extra processor time, but it guarantees that the list of users is up-to-date, regardless of the whims and crashes of the network.

The CheckForUsers routine is a bit complicated. The current list of other Chatterboxes in the zone is displayed using the list manager. Whenever CheckForUsers scans the network for other users, it must add new Chatterboxes to the list and delete Chatterboxes no longer active from the list - all without disturbing the user's selections in the list. It first calls NBPLookUp to get the names of all Chatterboxes, then it extracts one name at a time using NBPEXtract. Each name is compared to those in the list; if there's a match, that entry is marked as "found" and the next name is processed. If the name is not found in the list, it's added. After all of the names have been extracted, any names not marked as "found" are deleted from the list. This would all be better handled by a custom LDEF, but I'll save that for another day.

To receive messages with DDP, a program must first ask for them and provide a buffer to store them in. The AskForMessages routine takes care of this. It's very simple, just a single call to DDPRead. In this case, DDPRead is called with the async parameter set to true, meaning that the program will go on with its business while it waits for a message to come in. Once DDPRead has been called, it's the program's responsibility to watch the abResult field of the InABRecord parameter block to go to zero or a negative value. DDPRead sets this field to a positive number - the socket listener will set it to zero (if everything is OK) or negative (the error code) when a message comes in. In Chatterbox, the abResult field is examined once each time a null event is received.

When a message has been detected, the ReadMessage routine is called to pull the message out of the buffer and display it in the incoming messages field. As soon as the message is dealt with, ReadMessage calls AskForMessage to get ready for the next message.

Sending messages is simple too. When the user hits <return> to send a message, it's pulled out of the TextEdit field and put into a message buffer. The program then runs through the user list, building a parameter block for each user selected and calling DDPWrite to send the message. Sometimes the SendMessage routine must be called upon to send a message to everyone in the list, even if they aren't selected (this happens during the start-up greeting, for example). In this case, the calling routine sets the toAll parameter true.

When the user quits the program, it again sends out a special message that starts with a non-breaking space. This time it's an "adios" message. This message, like the greeting message, is used to tell all of the other Chatterbox programs in the zone to rebuild their lists of users, since one has just quit.

Extras and Options

That's the bulk of the program, but there are a few extras. There are two buttons in the window for frequently-used tasks. The Scan button just looks for Chatterboxes on the network. The user can press it if things seem a little screwy or out of sync - it's

really just a security blanket, it's rarely needed. The Select All button selects all of the users in the list, something I found most users do often. There are also three optional behaviors that can be set or unset in the Options menu. The first sets a flag that tells the program to beep with each incoming message. This is in case a message comes in when the user isn't watching or the window is in the background under MultiFinder. The second option allows the user to see the messages transmitted echoed in the incoming message field. The last option allows the user to have all new users that come along be automatically selected as recipients.

What next?

So how could this program be improved? Well, it would be nice if the program let you pick the zone you wanted to chat in, instead of just letting you chat in your own. The *AppleTalk Zone Information Protocol* could be used to get a list of zones on the internet. It would also be nice if the incoming messages that were deleted from the TextEdit field as they scrolled off the top could instead be scrolled back down with a scroll bar. A file transfer facility would be nice. No doubt all of you out there can come up with more improvements.

As you see, AppleTalk is easy to use. This simple, low-level approach, using DDP and NBP, could be used to implement games, e-mail, network copy protection schemes and so on. As more and more Macs are networked out there, network awareness becomes more and more a must for any application.

Listing: Chatter.c

```
/* ===== Chatterbox ===== */
```

```
#include <AppleTalk.h>
```

```
/** defines */
```

```
#define NULL 0L
#define MAXUSERS 64
#define BUFSIZE 600
#define NONE 0
#define REGISTER 1
#define WAITING 2
#define SCANNING 3
#define SENDING 4
#define RECEIVING 5
#define MAXTEXT 32000L
#define MAXMSG 50
#define DISPLINES 8
#define RETURNKEY '\015'
#define ENTERKEY '\003'
```

```
/** data about a single user */
```

```
typedef struct UserRecord {
    AddrBlock addr;
    int found;
} UserRecord;
```

```
/** Globals */
```

```
int DoneFlag;
WindowPtr TheWindow;
ControlHandle ScanButton;
ControlHandle SelectAllButton;
ListHandle UserList;
TEHandle MyMessage;
TEHandle InMessages;
```

```
MenuHandle AppleMenu;
MenuHandle FileMenu;
MenuHandle EditMenu;
MenuHandle OptionMenu;
DDPProto **InABRecord;
UserRecord OtherUsers[MAXUSERS];
EntityName MyEntityName;
Ptr MyNameBuf;
Str255 MyName;
int MySocketID;
int NumOtherUsers;
int TheError;
int EchoFlag;
int BeepFlag;
int SelectFlag;
char RecBuffer[BUFSIZE];
Rect StatusR;
Rect DragRect;
```

```
/*=====*/
```

```
main()
```

```
{
    EventRecord event;
    WindowPtr whichWindow;
    char key;
```

```
Initialize();
```

```
while ( !DoneFlag ) {
```

```
    GetNextEvent ( everyEvent, &event );
```

```
    switch ( event.what ) {
```

```
        case activateEvt:
```

```
            whichWindow = (WindowPtr)event.message;
```

```
            if ( whichWindow == TheWindow ) {
```

```
                if ( event.modifiers & activeFlag ) {
```

```
                    TEActivate ( MyMessage );
```

```
                    LActivate ( true, UserList );
```

```
                    ShowControl ( ScanButton );
```

```
                    ShowControl ( SelectAllButton );
```

```
                }
```

```
            } else {
```

```
                TEDeactivate ( MyMessage );
```

```
                LActivate ( false, UserList );
```

```
                HideControl ( ScanButton );
```

```
                HideControl ( SelectAllButton );
```

```
            }
```

```
        break;
```

```
        case updateEvt:
```

```
            whichWindow = (WindowPtr)event.message;
```

```
            if ( whichWindow == TheWindow ) {
```

```
                BeginUpdate ( TheWindow );
```

```
                DoUpdate ( );
```

```
                EndUpdate ( TheWindow );
```

```
            }
```

```
        break;
```

```
        case mouseDown:
```

```
            switch ( FindWindow ( event.where,
                                &whichWindow ) ) {
```

```
                case inMenuBar:
```

```
                    DoMenu ( MenuSelect(event.where));
```

```
                    break;
```

```
                case inSysWindow:
```

```
                    SystemClick ( &event, whichWindow );
```

```
                    break;
```

```
                case inDrag:
```

```
                    if ( whichWindow == TheWindow )
```

```
                        DragWindow ( TheWindow,
```

```
                        event.where, &DragRect );
```

```
                    break;
```

```
                case inContent:
```

```
                    if ( whichWindow == TheWindow ) {
```

```
                        if ( TheWindow != FrontWindow() )
```

```
                            SelectWindow ( TheWindow );
```

```

        else DoMouseDown ( event.where,
            event.modifiers );
    }
    break;
}
break;
case autoKey:
case keyDown:
    key = (char) (event.message & charCodeMask);
    if ( event.modifiers & cmdKey )
        DoMenu ( MenuKey ( key ) );
    else if ( key == RETURNKEY || key == ENTERKEY )
        SendMessage ( false );
    else {
        ForeColor ( greenColor );
        TEKey ( key, MyMessage );
        if ((*MyMessage)->teLength > MAXMSG)
            TEText ( (long) MAXMSG,
                MAXTEXT, MyMessage );
        TETDelete ( MyMessage );
    }
    ForeColor ( blackColor );
}
break;
case nullEvent:
    if ( MySocketID == 0 ) StartAT ( );
    else {
        ShowStatus ( WAITING );
        if ( (*InABRecord)->abResult <= 0 )
            ReadMessage ( );
        TEIdle ( MyMessage );
    }
    break;
}
}

```

```

ByeBye ( );
}

```

```

/*===== initialize the Mac and my globals */

```

```

Initialize ( )
{
    EventRecord  event;
    Handle       h;

    /*** init the mac rom stuff */
    InitGraf ( &thePort );
    InitFonts ( );
    InitWindows ( );
    InitMenus ( );
    TEInit ( );
    InitDialogs ( 0L );
    InitCursor ( );
    FlushEvents ( everyEvent, 0 );

    /*** set up a few globals ... */
    EchoFlag = BeepFlag = SelectFlag = true;
    DoneFlag = false;
    MyNameBuf = 0L;
    InABRecord = (DDPProto **)
        NewHandle ( sizeof ( DDPProto ) );
    MoveHI ( InABRecord );
    HLock ( InABRecord );
    DragRect = screenBits.bounds;
    MySocketID = 0;
    NumOtherUsers = 0;

    /*** read in the menus */
    AppleMenu = GetMenu ( 1000 );
    InsertMenu ( AppleMenu, 0 );
    AddResMenu ( AppleMenu, 'DRVr' );
    FileMenu = GetMenu ( 1001 );
    InsertMenu ( FileMenu, 0 );
    EditMenu = GetMenu ( 1002 );

```

```

    InsertMenu ( EditMenu, 0 );
    OptionMenu = GetMenu ( 1003 );
    InsertMenu ( OptionMenu, 0 );
    DrawMenuBar ( );

```

```

    /*** make sure appletalk is available */
    if ( (TheError = MPPOpen ( )) != noErr )
        ErrorAlert ( "\PCan't start AppleTalk, error:", TheError );

```

```

    /*** get user's name */
    h = (Handle) GetString ( -16096 );
    if ( h != NULL )
        BlockMove ( *h, MyName, ((long) **h) + 1L );
    else MyName[0] = 0;
    ReleaseResource ( h );
    NameDlog ( );

```

```

    /*** open our window & init */
    InitTheWindow ( );
}

```

```

/*===== process menu picks */

```

```

DoMenu ( mResult )
    long mResult;
    {
        int         theMenu, theItem, i;
        Str255      daname;
        DialogPtr    aboutdlog;
        GrafPtr      curport;
        TEHandle     te;

        theItem = LoWord ( mResult );
        theMenu = HiWord ( mResult );
        switch ( theMenu ) {
            case 1000:
                if ( theItem == 1 ) {
                    aboutdlog = GetNewDialog ( 2000, NULL, -1L );
                    i = 0;
                    while ( ! i ) ModalDialog ( NULL, &i );
                    DisposDialog ( aboutdlog );
                }
            else {
                GetItem ( AppleMenu, theItem, daname );
                GetPort ( &curport );
                OpenDeskAcc ( daname );
                SetPort ( curport );
            }
            break;
            case 1001:
                if ( theItem == 1 ) DoneFlag = true;
                break;
            case 1002:
                ForeColor ( greenColor );
                switch ( theItem ) {
                    case 3:
                        TECut ( MyMessage );
                        break;
                    case 4:
                        TECopy ( MyMessage );
                        break;
                    case 5:
                        TEPaste ( MyMessage );
                        if ((*MyMessage)->teLength > MAXMSG)
                            TETText ( (long) MAXMSG,
                                MAXTEXT, MyMessage );
                        TETDelete ( MyMessage );
                }
                break;
        }
        ForeColor ( blackColor );
        break;
        case 1003:
            switch ( theItem ) {
                case 1:

```

```

        if ( BeepFlag ) {
            BeepFlag = false;
            CheckItem (OptionMenu, 1, false );
        }
        else {
            BeepFlag = true;
            CheckItem (OptionMenu, 1, true );
        }
        break;
    case 2:
        if ( EchoFlag ) {
            EchoFlag = false;
            CheckItem (OptionMenu, 2, false );
        }
        else {
            EchoFlag = true;
            CheckItem (OptionMenu, 2, true );
        }
        break;
    case 3:
        if ( SelectFlag ) {
            SelectFlag = false;
            CheckItem (OptionMenu, 3, false );
        }
        else {
            SelectFlag = true;
            CheckItem (OptionMenu, 3, true );
        }
        break;
    }
    break;
}

HiliteMenu(0);
}

/*===== get the name the user wants to go by */
NameDialog ( )
{
    DialogPtr    dlog;
    int          itype, item;
    Handle       h, namefield;
    Rect         r;
    Str255       astr;

    /*** do the dialog */
    dlog = GetNewDialog ( 1000, 0L, -1L );
    GetDItem ( dlog, 4, &itype, &namefield, &r );
    SetIText ( namefield, MyName );
    SetIIText ( dlog, 4, 0, 255 );
    item = 0;
    while ( item != 1 ) ModalDialog ( 0L, &item );

    /*** save off the name, 32 chars max */
    GetIText ( namefield, &MyName );
    if ( ((int) *MyName) > 31 ) *MyName =(char) 31;

    CloseDialog ( dlog );
}

/*===== initialize AppleTalk stuff */
StartAT ( )
{
    /*** open a socket, use default listener */
    if (
        (TheError = DDPOpenSocket (&MySocketID, NULL)) != noErr )
        ErrorAlert ( "\PDDPOpenSocket ( ) returned:", TheError );

    RegisterName ( );
    CheckForUsers ( );
    SendGreeting ( );

    /*** listen for messages */
    AskForMessage ( );

```

```

    }

    /**== Register MyName as a "Chatterbox" via AppleTalk NBP */
    RegisterName ( )
    {
        int          len;
        NBPPProto    **myABRecNBP;

        /*** show some feedback */
        ShowStatus ( REGISTER );

        /*** make up a handle for registering */
        myABRecNBP = (NBPPProto **)
            NewHandle ( sizeof ( NBPPProto ) );
        HLock ( myABRecNBP );

        /*** set up our entity name */
        BlockMove ( MyName, MyEntityName.objStr,
            (long) *MyName) + 1L );
        BlockMove ( "\PChatterbox",
            MyEntityName.typeStr, 11L );
        BlockMove ( "\P*", MyEntityName.zoneStr, 2L );

        /*** set up a buffer for NBP's internal use */
        len = ((int) *MyName) + 26;
        MyNameBuf = NewPtr ( (long) len );

        /*** fill in the record & register */
        (*myABRecNBP)->abOpcode = tNBPRegister;
        (*myABRecNBP)->abUserReference = 0L;
        (*myABRecNBP)->nbpEntityPtr = &MyEntityName;
        (*myABRecNBP)->nbpBufPtr = MyNameBuf;
        (*myABRecNBP)->nbpBufSize = len;
        (*myABRecNBP)->nbpAddress.aSocket = (Byte)
            MySocketID;
        (*myABRecNBP)->nbpRetransmitInfo.
            retransInterval = 8;
        (*myABRecNBP)->nbpRetransmitInfo.retransCount = 2;
        if ( (TheError = NBPRegister(myABRecNBP, false)) != noErr )
            ErrorAlert ( "\PNBPRegister ( ) returned:", TheError );

        HUnlock ( myABRecNBP );
        DisposHandle ( myABRecNBP );
    }

    /**===== remove the name from NBP */
    UnRegisterName ( )
    {
        if ( (TheError = NBPRemove(&MyEntityName)) != noErr )
            ErrorAlert ( "\PNBPRemove ( ) returned:", TheError );
        DisposPtr ( MyNameBuf );
    }

    /**===== check for other users on the net. */
    CheckForUsers ( )
    {
        int          j, r, i, numFound, found, removed;
        Point        c;
        EntityName    otherEntity;
        NBPPProto    **otherABRecNBP;
        AddrBlock     addblock;

        /*** show some feedback */
        ShowStatus ( SCANNING );

        /*** mark them all as "not found" */
        for ( i = 0; i < NumOtherUsers; ++i )
            OtherUsers[i].found = false;

        /*** now scan for other users */
        BlockMove ( "\P=", otherEntity.objStr, 2L );
        BlockMove ( "\PChatterbox", otherEntity.typeStr, 11L );
        BlockMove ( "\P*", otherEntity.zoneStr, 2L );
        otherABRecNBP = (NBPPProto **)

```

```

NewHandle ( sizeof ( NBPPProto ) );
HLock ( otherABRecNBP );
(*otherABRecNBP)->abOpcode = tNBPLookup;
(*otherABRecNBP)->abUserReference = 0L;
(*otherABRecNBP)->nbpEntityPtr = &otherEntity;
(*otherABRecNBP)->nbpBufPtr = NewPtr ( 4096 );
(*otherABRecNBP)->nbpBufSize = 4096;
(*otherABRecNBP)->nbpDataField = 40;
(*otherABRecNBP)->nbpRetransmitInfo.
    retransInterval = 8;
(*otherABRecNBP)->nbpRetransmitInfo.
    retransCount = 1;
if ( (TheError = NBPLookup(otherABRecNBP, false)) != noErr)
    ErrorAlert ( "\PNBPLookup ( ) returned:", TheError );

/** if we found any, check for them in the
    list, else empty the list */
numFound = (*otherABRecNBP)->nbpDataField;
if ( numFound <= 0 ) {
    j = (*UserList)->dataBounds.bottom;
    if ( j > 0 ) LDelRow ( j, 0, UserList );
    NumOtherUsers = 0;
}
else {
    for ( i = 0; i < numFound; ++i ) {

        /** extract it */
        if ( (TheError = NBPExtract (
            (*otherABRecNBP)->nbpBufPtr,
            numFound, i+1, &otherEntity,
            &addblock )) == noErr ) {

            /** see if it's in the list */
            found = false;
            for ( j = 0; j < NumOtherUsers; ++j ) {
                if ( OtherUsers[j].addr.aSocket ==
                    addblock.aSocket &&
                    OtherUsers[j].addr.aNode ==
                    addblock.aNode &&
                    OtherUsers[j].addr.aNet ==
                    addblock.aNet ) {
                    found = true;
                    break;
                }
            }

            /** if it is, mark it */
            if ( found ) OtherUsers[j].found=true;

            /**else add it, select it if auto-select is on */
            else {
                c.h = 0;
                c.v = NumOtherUsers;
                OtherUsers[NumOtherUsers].addr =
                    addblock;
                OtherUsers[NumOtherUsers].found =
                    true;
                LAddRow ( 1, NumOtherUsers, UserList );
                LSetCell ( &otherEntity.objStr[1],
                    (int) otherEntity.objStr[0], c, UserList );
                if ( SelectFlag )
                    LSetSelect ( true, c, UserList );
                ++NumOtherUsers;
            }

            /** don't overflow */
            if ( NumOtherUsers >= MAXUSERS ) break;
        }
    }

    /**delete any that were not found (they've logged off) */
    for ( j = NumOtherUsers-1; j >= 0; --j ) {
        if ( !OtherUsers[j].found ) {
            --NumOtherUsers;

```

```

        BlockMove ( &OtherUsers[j+1],
            &OtherUsers[j],
            (long) ((NumOtherUsers-j)
                *sizeof(UserRecord)) );
        LDelRow ( 1, j, UserList );
    }
}

DisposPtr ( (*otherABRecNBP)->nbpBufPtr );
HUnlock ( otherABRecNBP );
DisposHandle ( otherABRecNBP );
}

/*===== tell DDP we're waiting for a msg */
AskForMessage ( )
{
    (*InABRecord)->abOpcode = tDDPRead;
    (*InABRecord)->abResult = 0;
    (*InABRecord)->abUserReference = 0L;
    (*InABRecord)->ddpType = 5;
    (*InABRecord)->ddpSocket = MySocketID;
    (*InABRecord)->ddpReqCount = BUFSIZE;
    (*InABRecord)->ddpDataPtr = RecBuffer;
    if ( (TheError = DDPRead ( InABRecord, true,
        true )) != noErr )
        ErrorAlert ( "\PDDPRead ( ) returned:", TheError );
}

/*===== read a message from the socket */
ReadMessage ( )
{
    Rect    r;
    Cell    c;
    Str255  othername;
    int     i, namelen, nlines, fromnode;

    /** make sure there's really a message */
    if ( (*InABRecord)->ddpActCount > 0 ) {

        /** show some feedback */
        ShowStatus ( RECEIVING );

        /** check for users if it's a special
            message (ie, has option-space) */
        if ( RecBuffer[0] == ' ' ) CheckForUsers ();

        /** see who it's from */
        namelen = 0;
        fromnode = (*InABRecord)->ddpAddress.aNode;
        for ( i = 0; i < NumOtherUsers; ++i )
            if ( fromnode ==
                OtherUsers[i].addr.aNode ) break;
        if ( i < NumOtherUsers ) {
            c.h = 0;
            c.v = i;
            namelen = 32;
            LGetCell ( othername, &namelen, c, UserList );
        }

        /** put the message in the display */
        ForeColor ( blueColor );
        TEText ( MAXTEXT, MAXTEXT, InMessages );
        TEInsert ( RecBuffer, (long) (*InABRecord)->ddpActCount,
            InMessages );
        TEInsert ( " (", 2L, InMessages );
        TEInsert ( othername, (long) namelen, InMessages );
        TEInsert ( ")\015", 2L, InMessages );
        nlines = (*InMessages)->nLines;
        for ( i = DISPLINES; i < nlines; ++i )
            DeleteFirstLine ( InMessages );
        ForeColor ( blackColor );

        if ( BeepFlag ) SysBeep ( 1 );

```

```

    }
    else ErrorAlert ( "\PError in recieved packet.", 0 );

    AskForMessage ( );
}

/*===== send the message out. If toAll is true,
send to all, else send to selected. */
SendMessage ( toAll )
{
    int    toAll;
    {
        DDPProto    **outABRecord;
        Handle      thetext;
        int          i, nlines, textlen, msgSent;
        Cell         c;
        char          xmitBuffer[BUFSIZE];

        /** only if there is something to say */
        if ( (*MyMessage)->teLength <= 0 ) return;

        /** get a record */
        outABRecord = (DDPProto **)
            NewHandle ( sizeof ( DDPProto ) );
        HLock ( outABRecord );

        /** copy the text to the out buffer, empty
        TextEdit record */
        thetext = (Handle) TEGetText ( MyMessage );
        HLock ( thetext );
        textlen = (*MyMessage)->teLength;
        BlockMove ( *thetext, xmitBuffer, textlen );
        HUnlock ( thetext );

        /** send the message to appropriate users */
        for(msgSent=false,i=0; i<NumOtherUsers; ++i){
            c.h = 0;
            c.v = i;
            if ( toAll ||
                LGetSelect ( false, &c, UserList ) ){
                ShowStatus ( SENDING );
                (*outABRecord)->abOpcode = tDDPWrite;
                (*outABRecord)->abUserReference = 0L;
                (*outABRecord)->ddpType = 5;
                (*outABRecord)->ddpSocket = MySocketID;
                (*outABRecord)->ddpAddress =
                    OtherUsers[i].addr;
                (*outABRecord)->ddpDataPtr = xmitBuffer;
                (*outABRecord)->ddpReqCount = textlen;
                if ( (TheError = DDPWrite ( outABRecord,
                    false, false )) != noErr )
                    ErrorAlert ( "\PDDPWrite ( ) returned:", TheError );
                msgSent = true;
            }
        }

        if ( msgSent ) {
            TESSetSelect ( 0L, MAXTEXT, MyMessage );
            TEDelete ( MyMessage );
            TESSetSelect ( 0L, 0L, MyMessage );

            if ( EchoFlag ) {
                ForeColor ( blueColor );
                TESSetSelect ( MAXTEXT, MAXTEXT, InMessages );
                TEInsert ( "«", 1L, InMessages );
                TEInsert ( xmitBuffer, (long) textlen, InMessages );
                TEInsert ( "»\015", 2L, InMessages );
                nlines = (*InMessages)->nLines;
                for ( i = DISPLINES; i < nlines; ++i )
                    DeleteFirstLine ( InMessages );
                ForeColor ( blackColor );
            }
        }

        HUnlock ( outABRecord );

```

```

        DisposHandle ( outABRecord );
    }

    /*===== say hello to existing users */
    SendGreeting ( )
    {
        Str255    greeting;

        /** send log-on message (has option-space) */
        greeting[0] = '\0';
        Pstrcat ( greeting, "\P Hello Everyone!" );
        TESSetSelect ( 0L, 0L, MyMessage );
        TEInsert ( &greeting[1], (long) greeting[0], MyMessage );
        SendMessage ( true );
        TESSetSelect ( 0L, MAXTEXT, MyMessage );
        TEDelete ( MyMessage );
    }

    /*===== pick up our toys and go home */
    ByeBye ( )
    {
        Str255    adios;

        /** send logoff message (has option-space) */
        adios[0] = '\0';
        Pstrcat ( adios, "\P " );
        Pstrcat ( adios, MyName );
        Pstrcat ( adios, "\P is off!" );
        TESSetSelect ( 0L, 0L, MyMessage );
        TEInsert ( &adios[1], (long) adios[0], MyMessage );
        SendMessage ( true );

        /** close everything */
        UnRegisterName ( );
        DDPCloseSocket ( (Byte) MySocketID );
        CloseWindow ( TheWindow );
    }

    /*===== delete the first line from the given TCHandle */
    DeleteFirstLine ( te )
    {
        TCHandle te;
        {
            Handle h;
            int    charsinline = 0;
            char    *cp;

            h = TEGetText ( te );
            HLock ( h );
            cp = *h;
            while ( *cp != '\015' ) {
                ++cp;
                ++charsinline;
            }
            ++charsinline;
            TESSetSelect ( 0L, (long) charsinline, te );
            TEDelete ( te );
        }
    }

    /*===== open and initialize the window */
    InitTheWindow ( )
    {
        Rect      r, dbr, view, dest;
        Point     csize;
        char      title[80];

        TheWindow = GetNewWindow ( 1000, 0L, -1L );
        title[0] = '\0';
        Pstrcat ( title, "\PChatterbox - " );
        Pstrcat ( title, MyName );
        SetWTitle ( TheWindow, title );
        SetPort ( TheWindow );
        TextFont ( 4 );
        TextSize ( 9 );
        TextFace ( 0 );

```

```

/** add in the controls */
ScanButton = GetNewControl ( 1000, TheWindow );
SelectAllButton =
    GetNewControl ( 2000, TheWindow );

/** add in the user list */
SetRect ( &r, 20, 22, 164, 94 );
SetRect ( &dbr, 0, 0, 1, 0 );
csize.v = 12;
csize.h = 300;
UserList = LNew ( &r, &dbr, csize, 0,
    TheWindow, true, false, false, true );

/** add in the outgoing message field */
SetRect ( &view, 21, 114, 339, 128 );
SetRect ( &dest, 22, 115, 338, 270 );
MyMessage = TNew ( &dest, &view );
TESetSelect ( 0L, 0L, MyMessage );

/** set up the incoming message area */
SetRect ( &view, 21, 148, 339, 238 );
SetRect ( &dest, 22, 149, 338, 270 );
InMessages = TNew ( &dest, &view );
TESetSelect ( 0L, 0L, InMessages );

/** set up the status area */
SetRect ( &StatusR, 196, 21, 340, 65 );
}

/*==== handle update events for our main (and only) window */
DoUpdate ( )
{
    Rect r;
    RgnHandle vis;

    /** do the controls */
    SetPort ( TheWindow );
    DrawControls ( TheWindow );

    /** show the user list */
    r = (*UserList)->rView;
    InsetRect ( &r, -1, -1 );
    FrameRect ( &r );
    MoveTo ( r.left, r.top-4 );
    DrawString ( "\PSelect Recipients:" );
    vis = TheWindow->visRgn;
    HandToHand ( &vis );
    LUpdate ( vis, UserList );
    DisposHandle ( vis );

    /** show the MyMessage field */
    r = (*MyMessage)->viewRect;
    MoveTo ( r.left-1, r.top-6 );
    DrawString ( "\PType a message, <return> to send:" );
    EraseRect ( &r );
    InsetRect ( &r, -2, -2 );
    FrameRect ( &r );
    ForeColor ( greenColor );
    TEUpdate ( &r, MyMessage );
    ForeColor ( blackColor );

    /** show the incoming message field */
    r = (*InMessages)->viewRect;
    MoveTo ( r.left-1, r.top-6 );
    DrawString ( "\PReceived Messages:" );
    EraseRect ( &r );
    InsetRect ( &r, -2, -2 );
    FrameRect ( &r );
    ForeColor ( blueColor );
    TEUpdate ( &r, InMessages );
    ForeColor ( blackColor );

    /** show the status area */

```

```

MoveTo ( StatusR.left, StatusR.top-4 );
DrawString ( "\PStatus:" );
FrameRect ( &StatusR );
ShowStatus ( NONE );
}

/*==== display the status notice */
ShowStatus ( status )
int status;
{
    Rect r;
    char *cp;
    static int oldstatus;

    if ( status != oldstatus ) {
        InsetRect ( &StatusR, 2, 2 );
        EraseRect ( &StatusR );
        ForeColor ( redColor );
        cp = NULL;
        switch ( status ) {
            case NONE:
                break;
            case REGISTER:
                cp = "\PRegistering on the network.";
                break;
            case WAITING:
                cp = "\PListening...";
                break;
            case SCANNING:
                cp = "\PScanning for other users.";
                break;
            case SENDING:
                cp = "\PSending message.";
                break;
            case RECEIVING:
                cp = "\PReceiving message.";
                break;
        }
        if ( cp != NULL )
            TextBox ( (cp + 1), (long) *cp, &StatusR, teJustLeft );
        ForeColor ( blackColor );
        InsetRect ( &StatusR, -2, -2 );
        oldstatus = status;
    }
}

/*==== handle mouse-downs in our window */
DoMouseDown ( thePt, mods )
Point thePt;
int mods;
{
    Rect r;
    Point localPt, c;
    ControlHandle chan;

    /** get the local point */
    localPt = thePt;
    GlobalToLocal ( &localPt );

    /** in UserList or it's scroll bars? */
    r = (*UserList)->rView;
    r.right += 16;
    if ( PtInRect ( localPt, &r ) &&
        ( NumOtherUsers > 0 ) ) {
        LClick ( localPt, mods, UserList );
        return;
    }

    /** in MyMessage? */
    r = (*MyMessage)->viewRect;
    if ( PtInRect ( localPt, &r ) ) {
        TEClick ( localPt,
            ((mods & shiftKey) == shiftKey),
            MyMessage );
    }
}

```

```

return;
}

/** in the controls */
if (FindControl(localPt, TheWindow, &chan)){
    if (TrackControl ( chan, localPt, NULL ) ){
        if ( chan == ScanButton ) CheckForUsers();
        if ( chan == SelectAllButton ) {
            for ( c.h = 0, c.v = (*UserList)->dataBounds.top;
                c.v < (*UserList)->dataBounds.bottom; ++c.v )
                LSetSelect ( true, c, UserList );
        }
    }
}

/*==== catenate the second pascal string to the first */
Pstrcat ( str1, str2 )
char *str1, *str2;
{
    char *strllen, *strlend;

    strllen = str1;
    strlend = str1 + (int) *strllen + 1;
    BlockMove ((str2 + 1), strlend, (long) *str2);
    *strllen += (int) *str2;
}

/*== alert the user to an error, optionally exit to shell */
ErrorAlert ( where, errorNum )
char *where;
int errorNum;
{
    Str255 idstr;

    /** give the alert */
    NumToString ( (long) errorNum, idstr );
    ParamText ( where, idstr, "\P", "\P" );
    if ( Alert ( 666, 0L ) == 1 ) {
        UnRegisterName ( );
        DDPCLoseSocket ( (Byte) MySocketID );
        ExitToShell ( );
    }
}

```

Listing: Chatter.r

```

resource 'BNDL' (128) (
    'chat',
    0,
    ( /* array TypeArray: 2 elements */
        /* [1] */
        'ICN#',
        ( /* array IDArray: 2 elements */
            /* [1] */
            0, 128,
            /* [2] */
            1, 129
        ),
        /* [2] */
        'FREF',
        ( /* array IDArray: 2 elements */
            /* [1] */
            0, 128,
            /* [2] */
            1, 129
        )
    )
);

data 'chat' (0) (
    $"15 43 68 61 74 65 72 62 6F 78 20 31 2E 30 20"
    $"62 79 20 52 6F 62 0D";
)

```

```

resource 'ICN#' (128) (
    ( /* array: 2 elements */
        /* [1] */
        $"1F FF FF F8 21 FF FF FC 41 FF FF FE 81 FF FF FF"
        $"81 FF FF FF 81 FF C0 FF 89 FF 00 3F 99 FE 00 1F"
        $"89 FC 00 0F 81 F8 00 07 81 F8 00 07 81 F0 00 03"
        $"80 F1 9D C3 80 F0 00 03 80 70 00 03 80 71 DD 43"
        $"80 70 00 03 80 70 00 03 80 F1 D7 03 87 F0 00 03"
        $"81 F0 00 03 81 F1 EE C3 81 F0 00 07 81 F0 00 07"
        $"81 F0 00 0F 87 E0 00 1F 8F 80 00 7F 87 FF FF FF"
        $"81 FF FF FF 41 FF FF FE 21 FF FF FC 1F FF FF F8",
        /* [2] */
        $"1F FF FF F8 3F FF FF FC 7F FF FF FE FF FF FF FF"
        $"FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF"
        $"FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF"
        $"FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF"
        $"FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF"
        $"FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF"
        $"FF FF FF FF 7F FF FF FE 3F FF FF FC 1F FF FF F8"
    )
);

resource 'WIND' (1000, "Chatterbox") (
    46, 16, 300, 376,
    documentProc,
    visible,
    noGoAway,
    0x0,
    "Chatterbox"
);

resource 'MENU' (1000, "Apple") (
    1000,
    textMenuProc,
    0x7FFFFFFD,
    enabled,
    apple,
    ( /* array: 2 elements */
        /* [1] */
        "About Chatterbox...", noIcon, noKey, noMark, plain,
        /* [2] */
        "-", noIcon, noKey, noMark, plain
    )
);

resource 'MENU' (1001, "File") (
    1001,
    textMenuProc,
    allEnabled,
    enabled,
    "File",
    ( /* array: 1 elements */
        /* [1] */
        "Quit", noIcon, "Q", noMark, plain
    )
);

resource 'MENU' (1002, "Edit") (
    1002,
    textMenuProc,
    0x7FFFFFFC,
    enabled,
    "Edit",
    ( /* array: 5 elements */
        /* [1] */
        "Undo", noIcon, "Z", noMark, plain,
        /* [2] */
        "-", noIcon, noKey, noMark, plain,
        /* [3] */
        "Cut", noIcon, "X", noMark, plain,
        /* [4] */
        "Copy", noIcon, "C", noMark, plain,
        /* [5] */

```



```

    "Paste", noIcon, "V", noMark, plain
)
);

resource 'MENU' (1003, "Options") (
    1003,
    textMenuProc,
    allEnabled,
    enabled,
    "Options",
    ( /* array: 3 elements */
        /* [1] */
        "Beep On Receipt", noIcon, noKey, check, plain,
        /* [2] */
        "Echo My Transmissions", noIcon, noKey, check, plain,
        /* [3] */
        "Auto-select New Users", noIcon, noKey, check, plain
    )
);

resource 'ALRT' (666, "Error Alert") (
    (162, 90, 330, 398),
    666,
    ( /* array: 4 elements */
        /* [1] */
        OK, visible, sound1,
        /* [2] */
        OK, visible, sound1,
        /* [3] */
        OK, visible, sound1,
        /* [4] */
        OK, visible, sound1
    )
);

resource 'cicn' (1000) (
    16,
    (0, 0, 32, 32),
    0,
    unpacked,
    0,
    0x480000,
    0x480000,
    chunky,
    4,
    1,
    4,
    0,
    0,
    4,
    (0, 0, 32, 32),
    4,
    (0, 0, 32, 32),
    $*1F FF FF F8 3F FF FF FC 7F FF FF FE FF FF FF FF*
    $*FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF*
    $*FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF*
    $*FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF*
    $*FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF*
    $*FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF*
    $*FF FF FF FF 7F FF FF FE 3F FF FF FC 1F FF FF 8F*,
    $*1F FF FF F8 21 FF FF FC 41 FF FF FE 81 FF FF FF*,
    $*81 FF FF FF 81 FF C0 FF 89 FF 00 3F 99 FE 00 1F*,
    $*89 FC 00 0F 81 F8 00 07 81 F8 00 07 81 F0 00 03*,
    $*80 F1 DD C3 80 F0 00 03 80 70 00 03 80 71 DD 43",
    $*80 70 00 03 80 70 00 03 80 F1 D7 03 87 F0 00 03",
    $*81 F0 00 03 81 F1 EE C3 81 F0 00 07 81 F0 00 07",
    $*81 F0 00 0F 87 E0 00 1F 8F 80 00 7F 87 FF FF FF*,
    $*81 FF FF FF 41 FF FF FE 21 FF FF FC 1F FF FF F8",
    0x00, 0,
    ( /* array ColorSpec: 5 elements */
        /* [1] */
        0, 65535, 65535, 65535,
        /* [2] */

```

```

        1, 0, 0, 0,
        /* [3] */
        2, 8856, 65535, 6900,
        /* [4] */
        3, 7652, 8260, 65535,
        /* [5] */
        4, 65535, 9758, 8505
    ),
    $*00 01 11 11 11 11 11 11 11 11 11 11 11 10 00"
    $*00 10 00 01 33 33 33 33 33 33 33 33 33 31 00"
    $*01 00 00 01 33 33 33 33 33 33 33 33 33 33 10"
    $*10 00 00 01 33 33 33 33 33 33 33 33 33 33 31"
    $*10 00 00 01 33 33 33 33 33 33 33 33 33 33 31"
    $*10 00 10 01 33 33 33 33 00 00 00 00 00 33 31"
    $*10 01 10 01 33 33 33 30 00 00 00 00 00 03 33 31"
    $*10 00 10 01 33 33 33 00 00 00 00 00 00 33 31"
    $*10 00 00 01 33 33 33 30 00 00 00 00 00 03 31"
    $*10 00 00 01 33 33 30 00 00 00 00 00 00 03 31"
    $*10 00 00 01 33 33 00 00 00 00 00 00 00 03 31"
    $*10 00 00 00 13 33 00 04 44 04 44 04 44 00 03 31"
    $*10 00 00 00 13 33 00 00 00 00 00 00 00 00 31"
    $*10 00 00 00 01 33 00 00 00 00 00 00 00 00 31"
    $*10 00 00 00 01 33 00 04 44 40 44 04 04 00 03 31"
    $*10 00 00 00 01 33 00 00 00 00 00 00 00 00 31"
    $*10 00 00 00 13 33 00 04 44 04 04 44 00 00 31"
    $*10 00 01 11 33 33 00 00 00 00 00 00 00 00 31"
    $*10 00 00 01 33 33 00 00 00 00 00 00 00 00 31"
    $*10 00 00 01 33 33 00 04 44 40 44 40 44 00 03 31"
    $*10 00 00 01 33 33 00 00 00 00 00 00 00 00 31"
    $*10 00 00 01 33 33 00 00 00 00 00 00 00 00 31"
    $*10 00 00 01 33 33 00 00 00 00 00 00 00 00 31"
    $*10 00 01 13 33 30 00 00 00 00 00 00 00 00 33 31"
    $*10 00 13 33 30 00 00 00 00 00 00 00 00 03 33 31"
    $*10 00 01 13 33 33 33 33 33 33 33 33 33 33 31"
    $*10 00 00 01 33 33 33 33 33 33 33 33 33 33 31"
    $*01 00 00 01 33 33 33 33 33 33 33 33 33 33 10"
    $*00 10 00 01 33 33 33 33 33 33 33 33 33 33 00"
    $*00 01 11 11 11 11 11 11 11 11 11 11 11 10 00"
);

resource 'DITL' (1000) (
    ( /* array DITLarray: 4 elements */
        /* [1] */
        (80, 52, 100, 112),
        Button (
            enabled,
            "OK"
        ),
        /* [2] */
        (8, 8, 40, 40),
        Icon (
            disabled,
            1000
        ),
        /* [3] */
        (16, 52, 36, 288),
        StaticText (
            enabled,
            "Enter the name you want to go by:"
        ),
        /* [4] */
        (48, 56, 64, 256),
        EditText (
            enabled,
            ""
        )
    )
);

resource 'DITL' (666) (
    ( /* array DITLarray: 6 elements */
        /* [1] */

```

```

    {128, 16, 152, 128},
    Button (
        enabled,
        "Exit To Shell"
    ),
    /* [2] */
    {128, 168, 152, 296},
    Button (
        enabled,
        "OK, Keep Going"
    ),
    /* [3] */
    {8, 8, 40, 40},
    Icon (
        disabled,
        1000
    ),
    /* [4] */
    {16, 48, 40, 296},
    StaticText (
        disabled,
        "Oops! Something bad has happened:"
    ),
    /* [5] */
    {48, 48, 80, 296},
    StaticText (
        disabled,
        ""
    ),
    /* [6] */
    {88, 48, 112, 296},
    StaticText (
        disabled,
        ""
    )
}
};

```

```

resource 'DITL' (2000) (
    /* array DITLarray: 1 elements */
    /* [1] */
    {2, 8, 98, 313},
    Picture (
        enabled,
        2000
    )
}
);

```

```

resource 'ICON' (1000, purgeable) (
    $*1F FF FF F8 21 FF FF FC 41 FF FF FE 81 FF FF FF"
    $*81 FF FF FF 81 FF C0 FF 89 FF 00 3F 99 FE 00 1F"
    $*89 FC 00 0F 81 F8 00 07 81 F8 00 07 81 F0 00 03"
    $*80 F1 DD C3 80 F0 00 03 80 70 00 03 80 71 DD 43"
    $*80 70 00 03 80 70 00 03 80 F1 D7 03 87 F0 00 03"
    $*81 F0 00 03 81 F1 EE C3 81 F0 00 07 81 F0 00 07"
    $*81 F0 00 0F 87 E0 00 1F 8F 80 00 7F 87 FF FF FF"
    $*81 FF FF FF 41 FF FF FE 21 FF FF FC 1F FF FF F8"
);

```

```

resource 'DLOG' (1000, "Get Name") (
    {66, 40, 178, 360},
    altDBoxProc,
    visible,
    goAway,
    0x0,
    1000,
    "What's your name?"
);

```

```

resource 'DLOG' (2000, "About...") (
    {40, 40, 140, 360},
    dBoxProc,
    visible,

```

```

    goAway,
    0x0,
    2000,
    "New Dialog"
);

```

```

resource 'FREF' (128) (
    'APPL',
    0,
    ""
);

```

```

resource 'CNTL' (1000, "Scan") (
    {75, 274, 95, 340},
    0,
    visible,
    1,
    0,
    pushButProc,
    0,
    "Scan"
);

```

```

resource 'CNTL' (2000, "Select All") (
    {75, 196, 95, 264},
    0,
    visible,
    1,
    0,
    pushButProc,
    0,
    "Select All"
);

```

```

resource 'vers' (2) (
    0x1,
    0x0,
    0x0,
    0x0,
    verUs,
    "1.0",
    " by Rob Hafernik"
);

```

```

resource 'vers' (1) (
    0x1,
    0x0,
    0x0,
    0x0,
    verUs,
    "1.0",
    "1.0"
);

```

```

resource 'PICT' (2000) (
    {191, 81, 287, 386},
    VersionOne (
        /* array OpCodes: 4 elements */
        /* [1] */
        shortComment (
            130
        ),
        /* [2] */
        clipRgn (
            {0, 0, 720, 576},
            ""
        ),
        /* [3] */
        packBitsRect (
            40,
            {191, 80, 287, 392},
            {191, 81, 287, 386},
            {191, 81, 287, 386},
            srcCopy,

```

```

$*02 D9 00 02 D9 00 02 D9 00 08 00 01 F1 FF 00 80"
$*EB 00 09 01 02 1F F2 FF 00 C0 EB 00 09 01 04 1F"
$*F2 FF 00 E0 EB 00 09 01 08 1F F2 FF 00 F0 EB 00"
$*09 01 08 1F F2 FF 00 F0 EB 00 0B 02 08 1F FC F4"
$*00 01 0F F0 EB 00 0B 02 08 9F F0 F4 00 01 03 F0"
$*EB 00 0B 02 09 9F E0 F4 00 01 01 F0 EB 00 0A 02"
$*08 9F C0 F3 00 00 F0 EB 00 0A 02 08 1F 80 F3 00"
$*00 70 EB 00 0A 02 08 1F 80 F3 00 00 70 EB 00 26"
$*08 08 1F 00 7F C6 00 00 0C 0C FE 00 00 C0 FD 00"
$*0E 30 1F F8 00 00 1F F8 00 18 00 03 01 80 00 78"
$*FD 00 00 C6 FE 00 26 08 08 0F 00 C0 66 00 00 0C"
$*0C FE 00 00 C0 FD 00 0E 30 18 0C 00 00 18 0C 00"
$*18 00 03 01 80 00 C0 FD 00 00 06 FE 00 29 25 08"
$*0F 00 C0 07 F8 7F 1F 9F 8F E3 FE FF 0F E3 05 00"
$*30 18 0C C1 80 18 0C 7F 1F E0 03 01 8F E3 F8 FE"
$*3F EF F1 C6 18 FF 00 29 25 08 07 00 C0 06 0C C1"
$*8C 0C 18 33 80 C1 98 31 8C 00 30 18 0C C1 80 18"
$*0C C1 98 30 03 01 98 30 C1 83 30 18 C6 30 FF"
$*00 29 25 08 07 00 C0 06 0C 01 8C 0C 18 33 00 C1"
$*98 30 D8 00 30 1F F8 C1 80 1F F8 C1 98 30 03 FF"
$*80 30 C1 83 30 0C 18 C6 60 FF 00 29 25 08 07 00"
$*C0 06 0C 7F 8C 0C 1F F3 00 C1 98 30 70 00 30 18"
$*0C C1 80 18 C0 C1 98 30 03 01 8F F0 C1 FF 30 0C"
$*18 C7 C0 FF 00 29 25 08 07 00 C0 06 0C C1 8C 0C"
$*18 03 00 C1 98 30 70 00 30 18 0C C1 80 18 60 C1"
$*98 30 03 01 98 30 C1 80 30 0C 18 C6 60 FF 00 29"
$*25 08 0F 00 C0 06 0C C1 8C 0C 18 03 00 C1 98 30"
$*D8 00 30 18 0C C1 80 18 30 C1 98 30 03 01 98 30"
$*C1 80 30 0C 18 C6 30 FF 00 29 25 08 7F 00 C0 66"
$*0C C1 8C 0C 18 33 00 C1 98 31 8C 00 30 18 0C C1"
$*80 18 18 C1 98 30 03 01 98 30 C1 83 30 0C 18 C6"
$*18 FF 00 29 25 08 1F 00 7F C6 0C 7F 87 87 8F E3"
$*00 FF 0F E3 06 00 30 1F F8 7F 80 18 0C 7F 1F 0E"
$*03 01 8F F0 C0 FE 30 0C 18 C6 0C FF 00 0D 01 08"
$*1F F2 00 04 30 00 00 01 80 EF 00 0D 01 08 1F F2"
$*00 04 70 00 00 C1 80 EF 00 0C 01 08 1F F2 00 03"
$*70 00 00 7F EE 00 09 01 08 1F F2 00 00 F0 EB 00"
$*0A 01 08 7E F3 00 01 01 F0 EB 00 0A 01 08 F8 F3"
$*00 01 07 F0 EB 00 09 01 08 7F F2 FF 00 F0 EB 00"
$*09 01 08 1F F2 FF 00 F0 EB 00 09 01 04 1F F2 FF"
$*00 E0 EB 00 09 01 02 1F F2 FF 00 C0 EB 00 08 00"
$*01 F1 FF 00 80 EB 00 02 D9 00 02 D9 00 02 D9 00"
$*02 D9 00 02 D9 00 02 D9 00 02 D9 00 02 D9 00 02"
$*D9 00 02 D9 00 02 D9 00 02 D9 00 02 D9 00 02 D9"
$*00 0D 09 04 04 88 80 02 00 03 00 00 18 E3 00 0D"
$*09 04 04 88 80 02 00 01 00 00 20 E3 00 10 0C 04"
$*05 9E FC 03 F1 F1 3F 00 FA F3 E7 B9 E6 00 10 0C"
$*04 04 88 82 02 0A 09 20 80 23 04 14 44 E6 00 10"
$*0C 04 44 88 82 02 0A 09 20 80 22 04 14 44 E6 00"
$*10 0C 04 A4 88 82 02 0B F9 20 80 22 04 14 44 E6"
$*00 10 0C 05 14 88 82 02 0A 01 20 80 22 04 14 44"
$*E6 00 10 0C 06 0C 88 82 02 0A 09 20 80 22 04 14"
$*44 E6 00 10 0C 04 04 86 82 02 09 F1 3F 00 22 03"
$*E4 45 E6 00 06 FA 00 00 20 E1 00 06 FA 00 00 20"
$*E1 00 02 D9 00 02 D9 00 02 D9 00 02 D9 00 02 D9"
$*00 25 06 04 04 00 02 00 01 02 FE 00 0C 20 40 04"

```

```

$*40 00 00 20 20 00 10 00 0F E0 FC 00 03 01 FC 20"
$*08 FD 00 00 80 FE 00 25 06 06 0C 00 02 00 01 04"
$*FE 00 0C 20 80 04 40 00 00 30 60 00 10 00 08 10"
$*FC 00 03 01 02 00 08 FD 00 00 80 FE 00 28 24 05"
$*14 F9 7A 10 01 08 3E 5E 00 21 07 CF 7E 41 00 28"
$*A7 CB D0 80 08 0A 09 F2 F2 F1 F0 01 01 65 E8 40"
$*1F 2F 1F 80 FE 00 28 24 04 A5 05 82 20 01 10 41"
$*61 00 22 08 24 41 41 00 25 28 2C 11 00 08 0A 0A"
$*0B 0B 0A 08 01 01 26 08 80 20 B0 A0 80 FE 00 28"
$*17 04 44 05 02 40 01 F0 41 41 00 3E 00 24 41 41"
$*00 22 20 28 12 00 08 0A 08 FE 0A 09 08 01 01 24"
$*09 00 00 A0 A0 80 FE 00 28 24 04 04 FD 03 C0 01"
$*08 7F 41 00 21 07 E4 41 41 00 20 27 E8 1E 00 08"
$*0A 09 FA 0A 0B F8 01 01 24 0F 00 1F A0 A0 80 FE"
$*00 26 15 04 05 05 02 20 01 04 40 41 00 20 88 24"
$*41 41 00 20 28 28 11 00 08 FC 0A 09 00 01 01 24"
$*08 80 20 A0 A0 80 FE 00 28 17 04 05 05 02 10 01"
$*02 41 41 00 20 48 24 41 41 00 20 28 10 80 08"
$*12 1A FE 0A 09 08 01 02 24 08 40 20 A0 A0 80 FE"
$*00 28 24 04 04 FD 02 09 01 01 3E 41 20 20 27 E3"
$*41 3F 20 20 27 E8 10 48 0F E1 E9 FA 0A 09 F1 01"
$*FC 24 08 20 1F A0 9F 80 FE 00 17 FD 00 00 01 FD"
$*00 00 20 FD 00 01 01 20 FD 00 00 08 FB 00 00 01"
$*F5 00 17 FD 00 00 02 FD 00 00 40 FD 00 01 41 40"
$*FD 00 00 10 FB 00 00 02 F5 00 06 F3 00 00 3E E8"
$*00 02 D9 00 02 D9 00 02 D9 00 02 D9 00 1D 12 07"
$*FD 00 00 07 F8 00 00 80 00 10 1F F8 00 00 C0 00"
$*00 1C FD 00 03 30 30 00 03 F4 00 1D 12 00 41 00"
$*00 04 04 00 00 80 00 10 01 80 00 00 C0 00 00 0C"
$*FD 00 03 30 30 00 03 F4 00 20 1C 00 41 F8 F8 04"
$*04 F8 F9 E0 1F 3C 01 81 F8 FC FE 7F 1F 8C 7E 3F"
$*B0 C0 30 33 F3 FB 18 FC F6 00 20 1C 00 41 05 04"
$*04 05 05 04 80 20 90 01 83 0D 86 C3 71 B0 CC C3"
$*61 B0 C0 30 36 1B 83 31 86 F6 00 20 1C 00 41 05"
$*04 07 F9 05 00 80 00 90 01 83 0D 80 C3 61 B0 CC"
$*C3 61 B0 C0 33 36 1B 03 61 80 F6 00 20 1C 00 41"
$*05 FC 04 21 FC FC 80 1F 90 01 83 FD 80 C3 61 B0"
$*CC C3 61 B0 C0 37 B6 1B 03 E0 FE F6 00 20 1C 00"
$*41 05 00 04 11 00 04 80 20 90 01 83 01 80 C3 61"
$*B0 CC C3 61 B0 C0 3C F6 1B 03 30 06 F6 00 20 1C"
$*00 41 05 04 04 09 05 04 80 20 90 01 83 0D 86 C3"
$*61 B0 CC C3 61 B0 C0 38 76 1B 03 19 86 F6 00 20"
$*1C 00 41 04 F8 04 04 F8 F8 60 1F 8C 01 81 F8 FC"
$*C3 61 9F 8C 7E 3F 9F C0 30 33 F3 03 0C FC F6 00"
$*08 ED 00 02 01 80 C0 F0 00 08 ED 00 02 61 B0 C0"
$*F0 00 08 ED 00 02 3F 1F 80 F0 00 02 D9 00 02 D9"
$*00 02 D9 00"
),
/* [4] */
shortComment {
    131
}
}
};

```



Jörg's Folder

C++ Sample Application



Jörg Langowski
MacTutor Editorial Board
Volume 5, Number 12

"C++ Sample Application"

As I am writing this, MPW C++ is shipping; version 3.1b1 is available through APDA as of October 11. The following message could be found on Applelink:

"On Tuesday, October 3, 1989 Apple Computer, Inc announced MPW C++ v.3.1B1 and said MPW C++ was available for ordering immediately and would be shipping later in October. 'Later' is here NOW! MPW C++ v.3.1B1 started shipping on Wednesday, October 11, 1989!

To get your copy, call APDA at
1-800-282-2732 (U.S.)
1-800-637-0029 (Canada)
1-408-562-3910 (International)

ask for part number M0346LL/A. The price is \$175 and the package includes one Apple C++ Manual, three AT&T C++ manual (Product Reference, Library Manual, and Selected Readings), MacApp 2.0B9 preliminary C++ interfaces (so you can use MacApp from C++), and three 3.5" disks.

Tim Swihart

C++ Product Manager "

Thus, all people interested in C++ can now get their copies - and follow this tutorial.

The example that I prepared for you this month is derived from one of the samples on the Apple C++ disks. Apple's samples include a rudimentary application framework, some sort of a mini-MacApp, defined in the classes TApplication and TDocument. After last month's introduction to some essential features of C++, I'll show you this time how to use this framework to build a small application that opens and closes a window in which some text is displayed and handles one custom menu in addition to the Apple, File, and Edit menus.

Since the base classes, TApplication and TDocument, provide for MultiFinder support, our application will also be fully MultiFinder compatible. I am not reprinting the full TApplication and TDocument framework here, "for copyright reasons" - the real reason being, of course, that it would make this article about ten pages longer, and those of you who use C++ have those files, anyway. However, we'll take a short look at the main features of those two classes.

TApplication implements the basic behavior of a Macintosh application. This class provides, among other methods, the constructor for instantiating a new application object, and the public EventLoop routine:

```
class TApplication : HandleObject {
```

```
public:  
TApplication(void);  
void EventLoop(void);  
...etc. ...  
}
```

Note here that this class is derived from the superclass **HandleObject**; this is a special class particular to MPW C++, where space for the object is allocated through a handle, not a pointer, to prevent memory fragmentation. Another 'special' superclass is **PascalObject**, which is used to access class definitions in Object Pascal from C++, necessary for MacApp support. We'll discuss these classes in a later column.

Most methods in **TApplication** are *protected* so that they can only be accessed by derived classes. They include basic event handlers and initializers which are called before and after the main event loop. Those methods are declared *virtual* which means they don't have to be defined within the class **TApplication** itself, and run-time binding will be supported where necessary.

The behavior of our application will be completely determined by the way we re-define **TApplication**'s methods. Of the base class, we only need the header file **TApplication.h** to make its definitions available to our particular implementation; the code for **TApplication** can be kept in a separate object file or a library.

A simple program would define its own application class, say **TMacTutorApp**, and override some of the event handlers in **TApplication**. The main program then just consist of calls to two methods, the constructor and the event loop:

```
int main (void)  
{  
    gTheApplication = new TMacTutorApp;  
    if (gTheApplication == nil) return 0;  
    gTheApplication->EventLoop();  
    return 0;  
}
```

A complete Macintosh application in five lines of C++ code! (I don't count the braces). Of course, this simplicity is deceptive; all the work is done in the methods that determine the behavior of the event loop. Our application class, its associated document class, and the methods are implemented in listing 1; the header file that contains the definitions is shown in listing 2.

Our Application

Our application class re-defines **TApplication**'s constructor and six private methods. Listing 1 contains the actual code. Let's explain the methods as they are called when the program is executed.

When the application object is constructed, first the constructor of the base class, **TApplication**, is called. By default, this method initializes all the toolbox managers, determines whether there is enough memory and the system environment is OK to run the program, and does some other initializations. Then, **TMacTutorApp**'s constructor is called (listing 1); this method sets up the menu bar and creates one new document (**DoNew()**).

Any application created using the **TApplication** framework contains a list of documents, whose maximum number is determined by the constant *kMaxOpenDocuments* in our application's class definition (Listing 2). The actual handling of this document list is implemented in **TApplication** itself and need not concern us here. As long as the number of open documents is less than *kMaxOpenDocuments*, the *New*, and possibly *Open*, items in the File menu are enabled; if the maximum number is reached, they will be disabled. This behavior is laid out in the **AdjustMenus** method. That method is called once on every pass through the event loop (also defined in the base class).

Mouse downs (and other events) are automatically passed on to their respective handlers by **TApplication**. The routine that we need to override in our class definition to handle menu selections is **DoMenuCommand** (Listing 1). Here, the basic apple, File and Edit menu selection are treated in a more or less standard way; the fourth menu is our own addition and contains four items to choose from. When one is selected, that item will be checked while the others are unchecked (checking/ unchecking is done by **AdjustMenus**). Furthermore, the number of the selected item, as well as a corresponding string, are passed on to the open document. The document then knows which message to display in its window.

Our Document

The basic methods that are defined in the **TDocument** class deal with document display (i.e. window updating, growing/ zooming, activate/deactivate), editing (cut/paste, mouse down in content, key down), and file and print handling. All these methods do nothing by default; they need to be overridden in our document's class definition.

Our document is called - what else - a **TMacTutorDocument**, and the methods we redefine are the constructor, destructor, window draw (private) and update methods. The constructor creates a new window and assigns an initial message to be displayed. When you look at its code (Listing 1), you'll notice that the first line looks somewhat funny:

```
TMacTutorDocument::TMacTutorDocument
(short resID, StringPtr s) : (resID) { etc... }
```

This form of a function call is particular to C++ constructors. When a constructor is called, it will call the constructor of the base class first; this constructor might need another set of parameters. This parameter list is therefore given after the colon. In our case, we pass our window's resource ID to the base class constructor, which then creates a new window according to the resource information. Thereafter our own constructor code is called, which initializes the message string and makes the window visible.

The destructor will only hide our window; the actual dele-

tion of the object (**DisposeWindow**) is done in the base class, **TDocument**.

DoUpdate will call our definition of **DrawWindow**, embedded in calls to **BeginUpdate** and **EndUpdate**. **DrawWindow** itself, which displays the message string in the window, is private; all window re-drawing is handled through calls to **DoUpdate**.

Methods inside **TMacTutorApp** set and retrieve the selected menu item number in our document, and set the message string. We therefore need access to these variables, which are private to **TMacTutorDocument**. Such access is provided through the methods **SetDisplayString**, **GetItemSelected**, and **SetItemSelected**, which are defined inline in the header file (listing 3).

Taking all these definitions together, you have an object-oriented framework for a very simple application. You see how easy it is to expand this framework to your own needs, and how well-separated the different parts of an application are in an object-oriented environment like C++. Application setup, menu handling (proper to the application), and window handling (proper to the document), are clearly distinct, as are the basic behavior (laid down in the application framework) and the user-defined behavior (by overriding the basic methods).

I'll leave it at that for this month; next time we'll define our own family of objects that can be displayed and manipulated in a document window. If you have questions or comments regarding this column, interesting pieces of C++ code, or suggestions for improvement, feel free to contact me via MacTutor or through the network: LANGOWSKI@FREMBL51.BITNET.

Listing 1: MacTutorApp.cp - our application-specific class implementations

```
/*-----
 * MacTutorApp
 *
 * A rudimentary application skeleton
 * based on an example given by Apple MacDTS
 * © J. Langowski / MacTutor 1989
 *
 * This example uses the TApplication and TDocument
 * classes defined in the Apple C++ examples
 *-----*/
#include <Types.h>
#include <QuickDraw.h>
#include <Fonts.h>
#include <Events.h>
#include <OSEvents.h>
#include <Controls.h>
#include <Windows.h>
#include <Menus.h>
#include <TextEdit.h>
#include <Dialogs.h>
#include <Desk.h>
#include <Scrap.h>
#include <ToolUtils.h>
#include <Memory.h>
#include <SegLoad.h>
#include <Files.h>
#include <OSUtils.h>
#include <Traps.h>
#include <StdLib.h>
```

```

// Constants, resource definitions, etc.

#define kMinSize 48 // min heap needed in K

#define rMenuBar 128 /* application's menu bar */
#define rAboutAlert 128 /* about alert */
#define rDocWindow 128 /* application's window */

#define mApple 128 /* Apple menu */
#define iAbout 1

#define mFile 129 /* File menu */
#define iNew 1
#define iClose 4
#define iQuit 12

#define mEdit 130 /* Edit menu */
#define iUndo 1
#define iCut 3
#define iCopy 4
#define iPaste 5
#define iClear 6

#define myMenu 131 /* Sample menu */
#define item1 1
#define item2 2
#define item3 3
#define item5 5

#include "TDocument.h"
#include "TApplication.h"
#include "MacTutorApp.h"

// create and delete document windows
// call initializer of base class TDocument
// with our window resource ID
TMacTutorDocument::TMacTutorDocument(
    (short resID, StringPtr s) : (resID)
{
    SetDisplayString(s);
    ShowWindow(fDocWindow); // Make sure the window is visible
}

TMacTutorDocument::~TMacTutorDocument(void)
{
    HideWindow(fDocWindow);
}

void TMacTutorDocument::DoUpdate(void)
{
    BeginUpdate(fDocWindow); // this sets up the visRgn
    if ( ! EmptyRgn(fDocWindow->visRgn) )
        // draw if updating needs to be done
        {
            DrawWindow();
        }
    EndUpdate(fDocWindow);
}

// Draw the contents of an application window.
void TMacTutorDocument::DrawWindow(void)
{
    SetPort(fDocWindow);
    EraseRect(&fDocWindow->portRect);

    MoveTo(100, 100);
    TextSize(18); TextFont(monaco);
    DrawString(fDisplayString);
} // DrawWindow

// Methods for our application class
TMacTutorApp::TMacTutorApp(void)
{

```

```

    Handle menuBar;

    // read menus into menu bar
    menuBar = GetNewMBar(rMenuBar);
    // install menus
    SetMenuBar(menuBar);
    DisposHandle(menuBar);
    // add DA names to Apple menu
    AddResMenu(GetMHandle(mApple), 'DRVr');
    DrawMenuBar();

    // create empty mouse region for MouseMoved events
    fMouseRgn = NewRgn();
    // create a single empty document
    DoNew();
}

// Tell TApplication class how much heap we need
long TMacTutorApp::HeapNeeded(void)
{
    return (kMinSize * 1024);
}

// Calculate a sleep value for WaitNextEvent.
// method proposed in the Apple example

unsigned long TMacTutorApp::SleepVal(void)
{
    unsigned long sleep;
    const long kSleepTime = 0x7fffffff;
    sleep = kSleepTime; // default value for sleep
    if ( (!fInBackground) )
    {
        sleep = GetCaretTime();
        // A reasonable time interval for MenuClocks, etc.
    }
    return sleep;
}

void TMacTutorApp::AdjustMenus(void)
{
    WindowPtr frontmost;
    MenuHandle menu;
    Boolean undo, cutCopyClear, paste;

    TMacTutorDocument* fMacTutorCurDoc =
        (TMacTutorDocument*) fCurDoc;
    frontmost = FrontWindow();

    menu = GetMHandle(mFile);
    if ( fDocList->NumDocs() < kMaxOpenDocuments )
        EnableItem(menu, iNew); // New is enabled when we can
    // open more documents
    else DisableItem(menu, iNew);
    if ( frontmost != (WindowPtr) nil )
        // is there a window to close?
        EnableItem(menu, iClose);
    else DisableItem(menu, iClose);

    undo = false; cutCopyClear = false; paste = false;

    if ( fMacTutorCurDoc == nil )
    {
        undo = true; // all editing is enabled for DA windows
        cutCopyClear = true;
        paste = true;
    }

    menu = GetMHandle(mEdit);
    if ( undo ) EnableItem(menu, iUndo);
    else DisableItem(menu, iUndo);

    if ( cutCopyClear )
        EnableItem(menu, iCut);

```

```

    EnableItem(menu, iCopy);
    EnableItem(menu, iClear);
}
else
{
    DisableItem(menu, iCut);
    DisableItem(menu, iCopy);
    DisableItem(menu, iClear);
}

if ( paste ) EnableItem(menu, iPaste);
else DisableItem(menu, iPaste);

menu = GetMHandle(myMenu);
EnableItem(menu, item1);
EnableItem(menu, item2);
EnableItem(menu, item3);
EnableItem(menu, item5);

CheckItem(menu, item1, false);
CheckItem(menu, item2, false);
CheckItem(menu, item3, false);
CheckItem(menu, item5, false);
CheckItem
(menu, fMacTutorCurDoc->GetItemSelected(), true);
} // AdjustMenus

void TMacTutorApp::DoMenuCommand
    (short menuID, short menuItem)
{
    short    itemHit;
    Str255   daName;
    short    daRefNum;
    WindowPtr window;
    TMacTutorDocument* fMacTutorCurDoc =
        (TMacTutorDocument*) fCurDoc;
    window = FrontWindow();
    switch ( menuID )
    {
        case mApple:
            switch ( menuItem )
            {
                case iAbout: // About box
                    itemHit = Alert(rAboutAlert, nil); break;
                default: // DAs etc.
                    GetItem(GetMHandle(mApple), menuItem, daName);
                    daRefNum = OpenDeskAcc(daName); break;
            }
            break;
        case mFile:
            switch ( menuItem )
            {
                case iNew: DoNew(); break;
                case iClose:
                    if (fMacTutorCurDoc != nil)
                    {
                        fDocList->RemoveDoc(fMacTutorCurDoc);
                        delete fMacTutorCurDoc;
                    }
                    else CloseDeskAcc
                        (((WindowPeek) fWhichWindow)->windowKind);
                    break;
                case iQuit: Terminate(); break;
            }
            break;
        case mEdit: // call SystemEdit for DA editing & MultiFin-
            der
            if ( !SystemEdit(menuItem-1) )
            {
                switch ( menuItem )
                {
                    case iCut: break;
                    case iCopy: break;
                    case iPaste: break;
                    case iClear: break;
                }
            }
        }
    }

```

```

    }
    break;
case myMenu:
    if (fMacTutorCurDoc != nil)
    {
        switch ( menuItem )
        {
            case item1:
                fMacTutorCurDoc->SetDisplayString("\pC++");
                break;
            case item2:
                fMacTutorCurDoc->SetDisplayString("\pSample");
                break;
            case item3:
                fMacTutorCurDoc->SetDisplayString("\pApplication");
                break;
            case item5:
                fMacTutorCurDoc->SetDisplayString("\pHave Fun");
                break;
        }
        fMacTutorCurDoc->SetItemSelected(menuItem);
        InvalRect(&window->portRect);
        fMacTutorCurDoc->DoUpdate();
    }
    break;
}
HiliteMenu(0);
} // DoMenuCommand

// Create a new document and window.
void TMacTutorApp::DoNew(void)
{
    TMacTutorDocument* tMacTutorDoc;
    tMacTutorDoc = new TMacTutorDocument
        (rDocWindow, "\pNothing selected yet.");
    // if we didn't get an allocation error, add it to list
    if (tMacTutorDoc != nil)
        fDocList->AddDoc(tMacTutorDoc);
} // DoNew

void TMacTutorApp::Terminate(void)
{
    ExitLoop(); // exits the main event loop
}

// Our application object, initialized in main().
TMacTutorApp *gTheApplication;

// main is the entrypoint to the program
int main(void)
{
    // Create our application object.
    // This also initializes the Toolbox -
    gTheApplication = new TMacTutorApp;
    if (gTheApplication == nil) // if we couldn't allocate
        object (impossible!?)
        return 0; // go back to Finder

    // Start main event loop
    gTheApplication->EventLoop();

    // return some value
    return 0;
}

```

Listing 2: MacTutorApp.h - class definitions

```

// Class definitions.

// Our document class.
// Only displays some text in a window
//
class TMacTutorDocument : public TDocument {

```



```

"Sample", noIcon, nokey, noMark, plain,
"Application", noIcon, nokey, noMark, plain,
"--", noIcon, noKey, noMark, plain,
"Have Fun", noIcon, nokey, noMark, plain
)
);

/* the About screen */
resource 'ALRT' (rAboutAlert, purgeable) (
    (40, 20, 190, 360), rAboutAlert, (
        OK, visible, silent;
        OK, visible, silent;
        OK, visible, silent;
        OK, visible, silent
    );
);

resource 'DITL' (rAboutAlert, purgeable) (
    (
        (120, 240, 140, 320),
        Button ( enabled, "OK" ),

        (8, 8, 24, 320),
        StaticText ( disabled,
            "MacTutorApp: C++ mini-application skeleton" ),

        (32, 8, 48, 320),
        StaticText ( disabled,
            "Copyright © 1989 J. Langowski / MacTutor" ),

        (56, 8, 72, 320),
        StaticText ( disabled,
            "[Based on examples by Apple MacDTSI" ),

        (80, 8, 112, 320),
        StaticText ( disabled,
            "Expand this application to your own taste" )
    )
);

resource 'WIND' (rDocWindow, preload, purgeable) (
    (64, 60, 314, 460),
    noGrowDocProc, invisible, goAway, 0x0,
    "MacTutor C++ demo"
);

resource 'STR#' (kMacTutorAppErrStrings, purgeable) (
    (
        "Not enough memory to run MacTutorApp";
        "Cannot create window";
    )
);

resource 'SIZE' (-1) (
    dontSaveScreen, acceptSuspendResumeEvents,
    enableOptionSwitch, canBackground,
    multiFinderAware, backgroundAndForeground,
    dontGetFrontClicks, ignoreChildDiedEvents,
    is32BitCompatible,
    reserved, reserved, reserved, reserved,
    reserved, reserved, reserved,
    kPrefSize * 1024, kMinSize * 1024
);

type 'JLMT' as 'STR';
resource 'JLMT' (0) (
    "MacTutor C++ Sample Application"
);

resource 'BNDL' (128) (
    'JLMT', 0,
    (
        'ICN#', ( 0, 128 ),
        'FREF', ( 0, 128 )
    )
);

```

```

)
);

resource 'FREF' (128) (
    'APPL', 0, ""
);

resource 'ICN#' (128) (
    ( /* MacTutor - JL ICN# */
        /* [1] */
        $"00 01 80 00 00 07 E0 00 00 1F F8 00 00 7F FE 00"
        $"01 FF FF 80 07 FF FF E0 0F FF 0F F8 07 FF 33 FC"
        $"03 FF FC 38 06 FF FF C8 0C 3F FF FE 08 0F FF D6"
        $"08 03 FF 96 08 F0 FF 19 09 F8 3E 16 09 88 0C 19"
        $"08 00 00 16 08 00 00 10 0B 1E 78 D0 08 FF FF D0"
        $"09 FF FF 90 FC 7E 7E 3E 96 00 00 6A D3 FF FF CA"
        $"52 00 00 4A 53 FF FF CB A6 38 70 69 DC 44 88 3F"
        $"1F 38 73 98 38 87 04 4C 67 08 83 86 7F FF FF FE",
        /* [2] */
        $"00 07 E0 00 00 1F F8 00 00 7F FE 00 01 FF FF 80"
        $"07 FF FF E0 1F FF FF F8 1F FF FF FC 0F FF FF FE"
        $"07 FF FF FC 07 FF FF F8 0F FF FF FE 0F FF FF FE"
        $"0F FF FF FE 0F FF FF FF 0F FF FF FE 0F FF FF FF"
        $"0F FF FF F6 0F FF FF F0 0F FF FF F0 0F FF FF F0"
        $"0F FF FF F0 0F FF FF FE F7 FF FF EE F3 FF FF CE"
        $"73 FF FF CE 73 FF FF CF E7 FF FF EF DF FF FF FF"
        $"1F FF FF F8 7F FF FF FE FF FF FF FF FF FF FF FF"
    )
);

```

Listing 4: MacTutorApp.make - the make file

```

# File:      MacTutorApp.make
# Target:    MacTutorApp
# Sources:   MacTutorApp.cp
#            MacTutorApp.h
#            MacTutorApp.r
#            TApplication.cp
#            TApplication.h
#            TDocument.cp
#            TDocument.h
#            TApplication.r
# Created:   Wednesday, October 18, 1989 8:15:31

OBJECTS = 0
MacTutorApp.cp.o TApplication.cp.o TDocument.cp.o

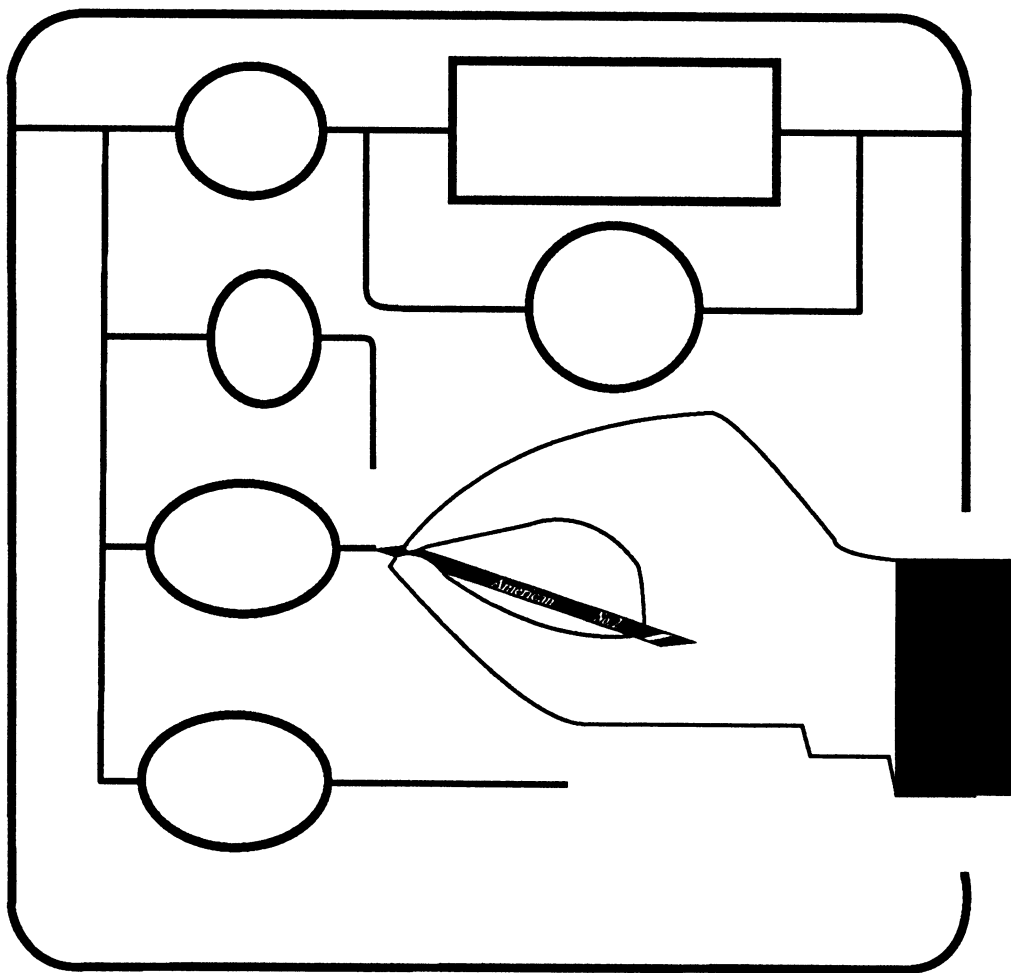
MacTutorApp.cp.o f 0
MacTutorApp.make MacTutorApp.cp MacTutorApp.h
CPlus MacTutorApp.cp
TApplication.cp.o f 0
MacTutorApp.make TApplication.cp TApplication.h
CPlus TApplication.cp
TDocument.cp.o f 0
MacTutorApp.make TDocument.cp TDocument.h
CPlus TDocument.cp

MacTutorApp ff MacTutorApp.make (OBJECTS)
Link -w -t APPL -c JLMT 0
    "(Libraries)"CRuntime.o 0
    (OBJECTS) 0
    "(Libraries)"Interface.o 0
    "(Libraries)"StdCLib.o 0
    "(Libraries)"CSANELib.o 0
    "(Libraries)"Math.o 0
    "(Libraries)"CInterface.o 0
    "(Libraries)"CPlusLib.o 0
    "(Libraries)"Complex.o 0
    -o MacTutorApp

MacTutorApp ff MacTutorApp.make MacTutorApp.r
Rez MacTutorApp.r -append -o MacTutorApp
MacTutorApp ff MacTutorApp.make TApplication.r
Rez TApplication.r -append -o MacTutorApp

```





Pascal Procedures

Pascal Procedures

Finder Icon Controls



Jean de Combret
Grenoble, France
Volume 5, Number 1

Finder-like Icon Controls

How to write, debug and use 'MDEF' and 'CDEF' resources.

Jean de Combret is a former river-hydraulics engineer, who worked with mathematical models human-unfriendly programs. He thinks the Mac could be a solution to make such programs more friendly and has therefore set up "Diadème Ingénierie" and written "ONDULA". He is a member of "Macintosh Alpes Club", as well as Didier Guillon, Jörg Langowski [hum... thanks! -JL] and a lot of other interesting folks.

Introduction

While developing my application, it appeared to me I needed an object-interface, somewhat like the Finder's one, with icons, hierarchy (folders contain other icons), actions (by double-clicking, classifying, throwing away or choosing in a menu). So I decided to write my own objects, and the best way I found to do it was to make them user-controls.

VarCodes

All icons of the family I've created bear the same appearance: an icon, and an underlying name. But each member of the family has different properties, which may be done by giving different "varCodes" to each variant. I choose a system of additive varCodes, although not all combinations have sense:

moveable=1;

means the control may be dragged inside its window (like all icons in the Finder);

doubleClickable=2;

means the control may generate an action by double-clicking it: it should then take an "open" appearance (like folders, disks and trash icons that generate a window, or like application icons that start-up applications);

trash=4;

means the control may "swallow up" another one released over its head. It should highlight while the mouse, dragging the control to be swallowed up, is still down and pointing to it (like trash, folders and disks icons in the Finder)

menu=8;

is a new type of control icon I've imagined: it should generate a pop-up menu of icons when pressed. This is an alternate technique to the palette menus and tear off menus. The menu is by the way bound to a window.

Some examples of possible combinations are listed below:

- a folder would be declared "movable + doubleClickable + trash" and its varCode would therefore be 7;
- a growing menu of icons would be "menu + trash" i.e. 12 ("menu" for removing or choosing items, "trash" to add items);

- an application icon would be declared "movable + doubleClickable" i.e. 3.

In fact, "menu" is exclusive of "movable" or "doubleClickable".

Menu definition procedure and pop-up menus

Before building our custom control definition procedure, we must first build our custom icons menus. At that point, the two first problems are

- to write a 'MDEF' code resource;
- to make it work with pop-up menus.

Inside Macintosh indicates p.I-362 how to write 'MDEF's for pull-down menus. In my alpha release of the fifth volume (for which the French Macintosh Development Support failed to send me updates), I don't find anything about pop-up menus nor about custom menus. I just found two significant pieces of information in the Color Menu Manager interface in my LightSpeed Pascal libraries -a new trap declared as

```
PROCEDURE PopUpMenuSelect (menu : MenuHandle;  
    top, left, popUpItem : INTEGER) : LONGINT;  
    inline $A80B;
```

and a constant :

```
mPopUpMsg=4;      ( sic )
```

This latter statement appeared later to be false. I learned by disassembling the standard code resource 'MDEF' 0 of the new system file, that only values ranging from 0 to 3 are accepted as messages, and that the one corresponding to pop-up menu treatments is 3:

```
mPopUpMsg=3;      ( right value )
```

By the way I learned that the routine responds to this new message by calculating "menuRect", given "menu", "top", "left" and "popUpItem" parameters of PopUpMenuSelect transmitted respectively through the "theMenu", "hitPt" and "whichItem" parameters from the menu definition procedure declaration :

```
PROCEDURE MyMenu (message : INTEGER;  
    theMenu : MenuHandle;  
    VAR menuRect : Rect;  
    hitPt : Point;  
    VAR whichItem : INTEGER);
```

Since I wrote most of this article, all of this has been confirmed by TechNote #172 from the November 1987 release. It just took 6 months to get it!

Before getting TechNote #172, I wasn't quite sure I hadn't made a mistake, but now I'm sure: hitPt is not a point! Instead it

receives "left" and "top" inverted from PopUpMenuSelect.

I also tried the trap with standard text menus. The two hints I shall give follow: (I found the second by decompiling the ROM patches; next time I'll first have a look to Inside Macintosh p.351!)

- pass -1 as beforeID parameter to InsertMenu (the same as for hierarchical menus) to indicate that theMenu is in the current menu list, but is not to be drawn in the menu bar.

- pass a positive non-zero integer as MenuID to NewMenu or nothing will ever appear on the screen (you may also store it in the 'MENU' resource read by GetMenu, of course).

Debugging A Code Resource (Part 1)

I shall indicate two techniques to test and debug code resources under LightSpeed Pascal. The first one, used here, is the roughest and should never be used for anything else than debugging. I learned the second one from the best Macintosh programming journal I know (I mean the April issue of MacTutor p.17 and p.30), and I should give an example of the use I made of it further in this article.

The problem is to access and debug the source text of a code resource while running a shell program that calls it via Menu Manager routines (or Control Manager routines for 'CDEF's). Normally the code resources is compiled separately and accessed through a handle to the 'MDEF' resource. The Menu Manager routines use to lock the 'MDEF' resource upon entry, dereference the handle, jump to its beginning and unlock it upon exit.

My technique consists in declaring the unit with the 'MDEF' source text in the uses statement of a small debugging-purpose shell program, and I create a new menu as follows :

```
myMenu := GetMenu(128);
myProcPtr := @main;
myMenu^.menuProc := @myProcPtr;
```

This way, the handle and its dereferenced pointer point to the stack, not at a masterpointer or a block within the heap. The HLock and HUnlock procedures may act on the high order bit of these pointers, it has no influence on the addresses (nevertheless be aware of TechNote #2 item 11). The Memory Manager never knows about the block and never tries to move it.

With this trick, you may freely use all the debugging facilities of LightSpeed Pascal: the Observe and LightsBug windows, the Step mode...

Once all the bugs inside the 'MDEF' code have been fixed (or at least seam to), you build your 'MDEF' code resource. I gave mine a ResID of 128. Under ResEdit, you can see that the compiler gave to this resource a default attribute "purgeable", you'd never ask it to give. Instead I changed it to "locked" and "preload", in order to have it always under the hand (a disk access while pulling a menu is not user-friendly) and to keep it in the low region of the application heap.

New Meaning of 'MENU' Resource

The 'MDEF' and 'CDEF' are not the only user resources I wrote for my application. I like the flexibility of building all sort of data resources as well as compiling code resources. In the current example, my menu consists of icons, described by their

names and their positions in the menu rectangle. It's slightly different from the standard 'MENU' resource information. But in order to use the Menu Manager routines, I preferred interpreting the same 'MENU' resources in a different way, rather than create a completely new resource type. Description of the resource format given p.I-364 of Inside Macintosh should be replaced by :

<u>Number of bytes</u>	<u>Contents</u>
2 bytes	MenuId
2 bytes	Placeholder for menu width
2 bytes	Placeholder for menu height
2 bytes	Resource ID of 'MDEF'
2 bytes	Place holder for 'MDEF' handle
4 bytes	EnableFlags
n+1 bytes	Title as Pascal string

for each menu item :

m+1 bytes	Text of item as Pascal string
2 bytes	Vertical coordinate of icon center
2 bytes	Horizontal coordinate of center
1 byte	0, indicating end of menu items (it is also a Pascal null string)

Note that this description occupies the same number of bytes that normal text menus do, which ensures the ability to use the standard procedures.

This description is almost the same as the one you can teach to ResEdit; I created a new 'TMPL' resource in the ResEdit file itself, which I gave the name 'MPOP'. However, in order to use the GetMenu routine, my menu resources shall not be of type 'MPOP' but of type 'MENU'. This is accomplished by creating a new 'MENU' resource and opening it with the "Open as..." command to indicate we wish to use the 'MPOP' format instead of the 'MENU' one to edit our resources (shortcut for "Open as..." is shift-option-double-clicking).

The 'TMPL' resource describes the structure of the 'MPOP' resource, somewhat like a Pascal record type describes the structure of a Pascal record variable (for more details see the MPW chapter about ResEdit) :

<u>Label</u>	<u>Type</u>
MenuID	DWRD
width	DWRD
height	DWRD
procID	DWRD
filler	DWRD
enableFlags	HLNG
title	PSTR
*****	LSTZ
menuItem	PSTR
center.v	DWRD
center.h	DWRD
*****	LSTE

Reading A Resource

In fact this new syntax allowed by ResEdit gives more generality and compactness than Pascal records do: strings in Pascal records always occupy 256 bytes in memory, even if only 10 are actually used. Consequently, Pascal is not able to decipher such resources. So I had to write some inline assembly routines for this purpose. These routines read a chosen type beginning at a given address that might be odd and return a Pascal formatted value on the stack. For strings it returns a StringHandle on the stack, pointing to a string of the actual size in the heap. I give here three of these inline routines :

```
FUNCTION GetNextByte
  (VAR LongAddress : LONGINT) : BYTE;
```

```
FUNCTION GetNextInteger
  (VAR LongAddress : LONGINT) : INTEGER;
```

```
FUNCTION GetNextString
  (VAR LongAddress : LONGINT) : StringHandle;
```

These functions return NIL if allocation failed.

For each of these routines LongAddress is incremented by the amount of bytes read.

The same routines in Pascal would have called BlockMove, which is very efficient for larger blocks but far too long for such a few bytes.

As an example, here is what GetNextByte would look like in Pascal :

```
FUNCTION GetNextByte (VAR longAddress : LONGINT) : BYTE;
VAR result : BYTE;
BEGIN
  result := 0;
  BlockMove(POINTER(longAddress), POINTER(ORD(@result) + 1),
  1);
  GetNextByte := result;
  longAddress:= longAddress+ 1;
END;
```

I added some other Pascal utilities for easy reading and writing :

```
FUNCTION SkipNextString (VAR LongAddress : LONGINT) : BYTE;
```

```
PROCEDURE SkipBytes (VAR LongAddress : LONGINT; byteCount :
INTEGER);
```

Routines and Messages

Let's come to our central topic: the control definition procedure. First of all note that there are 9 messages to the 'CDEF' code instead of only 4 to the 'MDEF' code. Controls are more complex than menus and hence more versatile.

I find Inside Macintosh not systematic enough in describing the meanings of these messages and I don't know other information about them. So I decided to build my own complete table of what messages each routine of the Control Manager sends to the 'CDEF' code. I wrote a small program to help me in this task by installing a spy at the entry of the 'CDEF' code. Figure 1. shows the table resulting from this spying.

ROUTINE	MESSAGES	COMMENT
NewControl	initCntl drawCntl	
GetNewControl	initCntl drawCntl	
DisposeControl	calcCRgns dispCntl	
KillControls	calcCRgns dispCntl	
SetCTitle	calcCRgns drawCntl	
HideControl	calcCRgns	
ShowControl	drawCntl	
DrawControls	drawCntl	
HiliteControl	drawCntl	
FindControl	testCntl	
TrackControl (in an indicator)	testCntl	before MouseUp
	thumbCntl	"
	dragCntl	"
	calcCRgns	"
TrackControl (otherwise)	posCntl	after MouseUp
	drawCntl	"
	testCntl	initialisation
	drawCntl	"
TrackControl (otherwise)	autoTrack	loop
	testCntl	"
	autoTrack	after MouseUp
TrackControl (otherwise)	drawCntl	"
	testCntl	
	drawCntl	
TestControl	testCntl	
MoveControl	calcCRgns drawCntl	
DragControl	dragCntl	before MouseUp
	calcCRgns	default method
	calcCRgns	after MouseUp
	drawCntl	(=MoveControl)
SizeControl	calcCRgns drawCntl	
SetCtlValue	drawCntl	
SetCtlMin	drawCntl	
SetCtlMax	drawCntl	

Fig. 1 : messages sent by the Control Manager routines

The program I used is listed below and I shall give here some explanations about the way I installed my spy, because it involves the second technique of building 'CDEF' resources from the same source file than the host program.

Debugging a Code Resource (Part 2)

Here comes the clean way of doing it (Thanks to Larry Rosenstein and also to Don Melton and Mike Ritter for the basic idea). You create a 6-byte handle and put a JMP \$xxxxxx

instruction in the block, which jumps to your procedure. Then you declare this block as a 'CDEF' resource which you give an unused ID. Then you can pass this ID to NewControl :

```

TYPE
  CDEFcodeHdl = ^CDEFcodePtr;
  CDEFcodePtr = ^CDEFcodeRecord;
  CDEFcodeRecord = RECORD
    jump : integer;
    address : ProcPtr;
  END;

VAR myControl : ControlHandle;
    myCDEF : CDEFcodeHdl; myCDEFid : integer;

FUNCTION ControlDefProc (varCode : INTEGER;
  theControl : ControlHandle; message : INTEGER;
  param : LONGINT) : LONGINT;
BEGIN
  ( put here the source of the 'CDEF' )
END;

BEGIN ( here begins your host program )
  myCDEF := CDEFcodeHdl (NewHandle(6));
  myCDEF^.jump := $4EF9;
  myCDEF^.address := @ControlDefProc;
  myCDEFid := UniqueID('CDEF');
  AddResource(handle(myCDEF), 'CDEF', myCDEFid, '');
  ( ***** )
  ( write here the inits of your host program )
  ( ***** )
  myControl := NewControl(theWindow, boundsRect, title,
  visible, value, min, max, myCDEFid * 16 + varCode, refCon);
  ( ***** )
  ( write here the body of your host program )
  ( ***** )
  DisposeControl(myControl);
  RmveResource(handle(myCDEF));
  DisposHandle(handle(myCDEF));
  ( ***** )
  ( write here the disposals of your host program )
  ( ***** )
END.

```

For more information, you'll have to refer to the listing given below (e.g. : don't have a too large UniqueID, it must be less than maxint DIV 16 ...).

Another interesting technique is the way of using routines passed by address from Pascal. Although Inside Macintosh says p.I-78 "Only routines written in assembly language ... can actually call the routine designated by a pointer of type ProcPtr", it's very easy to write inline code to do the job within Pascal programs. Suppose you have this function declaration :

```

FUNCTION MyFunction (firstArg : firstType;
  secondArg : secondType;
  VAR thirdArg : thirdType) : resultType;

```

If you want to call it by address (given by @MyFunction, or by a handle if the function is compiled as a code resource) you must declare an inline glue function.

For a function passed by pointer it should be :

```

FUNCTION CallMyFunction (firstArg : firstType;
  secondArg : secondType;
  VAR thirdArg : thirdType
  MyFunctionAddress : ProcPtr) : resultType;
INLINE

```

```

$205F, (MOVE.L (A7)+,A0)
$4E90; (JSR (A0))

```

or, for a function passed by handle (after having locked it):

```

FUNCTION CallMyFunction (firstArg : firstType;
  secondArg : secondType;
  VAR thirdArg : thirdType
  MyFunctionResource : handle) : resultType;
INLINE
  $205F, (MOVE.L (A7)+,A0)
  $2050, (MOVE.L (A0),A0)
  $4E90; (JSR (A0))

```

As you can see the declaration is exactly the same as in the original function, except that we add one last parameter. This technique applies as well to procedures. The number of arguments doesn't matter, only matters the coherence between the two declarations.

Coming back to our example, you'll notice a supplementary line in the inline code of my program, because tracing the 'CDEF' resource with MacsBug showed that it relied on a function return value being previously cleared.

Implementing the Control Definition Procedure

My goal was to do as much work as possible within the 'CDEF' code, in order to simplify the end programmer's job of writing the host program.

Although "movable" controls should act through the DragControl routine, I grouped all actions of all the variety of controls under the TrackControl routine. I implement also custom initializations and disposals.

Here follows descriptions of :

- what the ControlRecord fields are used to ;
- how the control definition procedure responds to each message ;
- how to use the Control Manager routines to make my controls work correctly.

ControlRecord fields

- ContrITitle contains the name that is used to find the related resources such as 'ICN#' and 'MENU'.
- in order to speed up memory access to the icon and menu records related to the control, I store handles to them in the following structure :

```

TYPE
  DataHandle = ^DataPointer;
  DataPointer = ^DataRecord;
  DataRecord = RECORD
    theIcon : handle;
    theMenu : MenuHandle;
  END;

```

and I store the DataHandle in the ContrlData field.

- I use the ContrlRefCon field to store the values returned from the 'CDEF' routines :
 - the ControlHandle of the "trash" in case the current control has been put in it;
 - the choice made from the icon menu, as returned by PopUpMenuSelect.

- **ContrlHilite** is not used.
- I use 6 levels of **ContrlValue** ranging from **ContrlMin** = 0 to **ContrlMax** = 5. Rather than a continuous scale, they correspond to six different states of the controls (the “**ContrlHilite**” field could have been used for the same purpose):



This is the data of my icon (of type 'ICN#')



This is its mask.



RestState=0;
(same as the data on white background)



SelectState = 1;



OpenState = 2;



SelectOpenState = 3;

Technical note #55 describes how the Finder draws icons in that variety of ways. But in fact it's not the actual way the old and new Finders do it. I think my algorithm is closer to reality. I've studied it using a funny feature of MacsBug -the Step Spy command :

SS 400000 400000

that slows all 68xxx instructions, and is specially recommended to examine graphical aspects of an application.

'CDEF' Routines In Response to the Messages

- **drawCntl** : draws the control with its title and its icon according to the state (i.e. its **ContrlValue**) ;
- **testCntl** : tests if the point is inside the icon's data or mask ;
- **calcCRgns** : calculates the region, union of the rectangles enclosing the icon and the title. This region is used for any erasing of the control and updating of the underlying pixels, or for drawing the dotted outline when dragging the control ;
- **initCntl** : stores the handles to the icon and to the menu in a structure, pointed to by another handle which is in turn stored in the **ContrlData** field, and sets the **actionProc** parameter to **POINTER(-1)**, to indicate we implement a custom **autoTrack** routine ;
- **dispCntl** : disposes the **ContrlData** handle but not the icon and menu handles which are resources and may be shared by

other controls ;

- **posCntl** : not used, because my controls don't have any indicator ;
- **thumbCntl** : not used ;
- **dragCntl** or **autoTrack** : the most important routine. First it sets the control to a selected state and deselects any other currently selected control. Then, if the control is “movable”, it detects the movements of the mouse, pulls a dotted outline and highlights the trashes the control flies over. If it was actually moved, it invalidates regions to be updated. If it was released over a “trash”, its value is set to 4 and the trash's **ControlHandle** is returned in **ContrlRefCon**. Then it looks for a double-click in case the control is “double-clickable” and hasn't been moved. In the other case where the control is a “menu” control, the routine generates the pop-up menu, by calling **PopUpMenuSelect** as described above and returns the choice in **ContrlRefCon**.

Note that a small problem arises in these routines when they call in turn other routines of the Control Manager -when the latter returns, the 'CDEF' resource is unlocked by the Control Manager, although the next instructions are still in the 'CDEF'. So it is imperative to re-lock the 'CDEF' resource after each call to the Control Manager within the 'CDEF'. If Apple had used **HGetSate** / **HSetState** instead of **HLock** / **HUnLock** to bracket the calls to 'CDEF', I think this problem would not exist.

Note also I let the user access to this last routine by two ways : **TrackControl**, which is more logic, and **DragControl** that calls “**drawCntl**” only when necessary, as shows figure 1., and thus makes a cleaner interface.

Control Manager Routines User's Guide

Here is the way the standard routines act on my custom controls:

- **NewControl** or **GetNewControl** are used to create the controls with the following values of the parameters :
 - **boundsRect** set to the 32x32 icon's rectangle ;
 - **max** = 5 ;
 - **min** = 0 ;
 - **procID** = 128 * 16 + **varCode** ;
 - **title** = 'CNTL' resource name
= 'ICN#' resource name
= 'MENU' resource name ;
- **TrackControl** is used to make the controls work properly. Pass **POINTER(-1)** as **actionProc**, in order to let the Control Manager generate **autoTrack** messages.
- **DragControl** is used for the same purpose (see above). Pass anything as **bounds**, **slop** and **axis**.
- **GetCRefCon** is used to get the choice returned from the pop-up menu or the handle to the “trash” the current control was put in ;
- **SetCtlValue** is used to close icons :
SetCtlValue(GetCtlValue(theOpenControl)-2);
or to deselect icons (such as a “trash”) :
SetCtlValue(GetCtlValue(theSelectedControl)-1);
- **HiliteControl** doesn't change anything;
- **SizeControl** only moves the gravity center of the control;

- never use SetCtlMin and SetCtlMax;
- the other routines (MoveControl, SetCtlTitle, HideControl, ShowControl) work the way you might expect.

Example Listing

I give hereafter the listing of the 'MDEF' and the 'CDEF' resources in LightSpeed Pascal as well as a minimum shell program as an example of how to use the custom controls. I apologize for the roughness of its interface: no standard menus, no DA's...It shows at least the simplicity for the end programmer.

Future Extensions

A feature found in the Finder and that I didn't implement is the ability of editing the title of the icon. It would probably need to define two parts in the control.

In some applications, pop-up menus change their title accordingly to the current choice. It could be possible here at the condition that the menu handle of the control is left unchanged, even though the title or the icon handle are changed.

At last, what would be "vachement classe" and truly up-to-date custom Mac programming is a color-hierarchical-pop-up-icon-user-menu !

Please send me a copy once you've written this marvel, or simply send me bug-reports and comments to:

Jean de Combret
Diadème Ingénierie
25, avenue de Constantine
38100 GRENOBLE
France

Listing 1: Control manager example

```
(*****)
( SPYING THE CONTROL MANAGER'S MESSAGES TO THE CDEF )
(*****)

( Put this file in the project after MacPasLib and MacTraps. )
( Don't forget "Use resource file" in "Run options" of menu )
( This resource file may be empty, or may contain a copy of )
( system file resources CDEF 0 and 1 )
(*****)
```

PROGRAM Controls;

CONST

CDEFID = 1; (0 for button, 1 for scroll bar)

TYPE

CDEFcodeHdl = ^CDEFcodePtr;
CDEFcodePtr = ^CDEFcodeRecord;
CDEFcodeRecord = RECORD
 jump : integer;
 address : ProcPtr;

END;

VAR

boundsRect, TextRect : rect;
myControl, whichControl : ControlHandle;
theEvent : eventRecord;
thePoint : point;
whichPart : integer;
theWindow, whichWindow : WindowPtr;
myFakeCDEF : CDEFcodeHdl;
myFakeCDEFid : integer;
CDEFProcHandle : handle;

LABEL

1, 2;

FUNCTION DoCDEF (varCode : integer; theControl : ControlHandle; message : integer; param : longint; ProcHandle : handle) : longint;

INLINE

(CDEF relies on prepared default value of 0 as funct. result:)

\$42AF, \$0010, (CLR.L 10(A7))

(JSR to a procedure passed by handle as last argument)

\$205F, (MOVE.L (A7)+,A0)

\$2050, (MOVE.L (A0),A0)

\$4E90; (JSR (A0))

FUNCTION ControlProc (varCode : integer; theControl : ControlHandle; message : integer; param : longint) : longint;

BEGIN

writeln(' message= ', message, ' param= ', param);

ControlProc := DoCDEF(varCode, theControl, message, param, CDEFProcHandle);

END;

BEGIN

(create a intermediate CDEF resource :)

myFakeCDEF :=

CDEFcodeHdl(NewHandle(sizeof(CDEFcodeRecord)));

IF MemError <> NoErr THEN

GOTO 2;

myFakeCDEF^.jump := \$4EF9;

myFakeCDEF^.address := @ControlProc;

REPEAT

myFakeCDEFid := UniqueID('CDEF');

(in order to have 16*myFakeCDEFid < maxint :)

UNTIL myFakeCDEFid < 1000;

AddResource(handle(myFakeCDEF), 'CDEF', myFakeCDEFid, '');

IF ResError <> NoErr THEN

GOTO 2;

SetRect(TextRect, 250, 40, 500, 330);

SetTextRect(TextRect);

ShowText;

SetRect(boundsRect, 40, 40, 200, 200);

theWindow := NewWindow(NIL, boundsrect, 'my window', true, 0, pointer(-1), false, 0);

SetPort(theWindow);

CDEFProcHandle := GetResource('CDEF', CDEFID);

IF ResError <> NoErr THEN

GOTO 1;

HLock(CDEFProcHandle);

IF MemError <> NoErr THEN

GOTO 1;

SetRect(boundsRect, 10, 10, 90, 26);

(now we begin to test the Control Manager's routines :)

writeln('NewControl : ');

myControl := NewControl(thePort, boundsRect, 'my control', true, 0, 0, 48, myFakeCDEFid * 16, 0);

SetCtlAction(myControl, pointer(-1));

writeln('SetCtlTitle : '); SetCtlTitle(myControl, 'new name');

writeln('HideControl : '); HideControl(myControl);

writeln('ShowControl : '); ShowControl(myControl);

writeln('HiliteControl : '); HiliteControl(myControl, 1);

writeln('HiliteControl : '); HiliteControl(myControl, 255);

writeln('HiliteControl : '); HiliteControl(myControl, 0);

writeln('SetCtlValue : '); SetCtlValue(myControl, 1);

writeln('SetCtlMax : '); SetCtlMax(myControl, 0);

writeln('SetCtlMax : '); SetCtlMax(myControl, 1);

writeln('SetCtlMin : '); SetCtlMin(myControl, 1);

writeln('SetCtlMin : '); SetCtlMin(myControl, 0);

writeln('SetCtlValue : '); SetCtlValue(myControl, 0);


```

writeln('MoveControl : '); MoveControl(myControl, 20, 20);
writeln('SizeControl : '); SizeControl(myControl, 100, 30);

( let's have loop to test "TrackControl" and "FindControl": )
writeln('The user should try actions on the control. ');
writeln('End by clicking outside the control. ');
FlushEvents(EveryEvent, 0); InitCursor;
REPEAT
  REPEAT
    UNTIL GetNextEvent(MDownMask, theEvent);
    thePoint := theEvent.where;
    whichPart := FindWindow(thePoint, whichWindow);
    SetPort(whichWindow); GlobalToLocal(thePoint);
    writeln('FindControl : ');
    whichPart :=
    FindControl(thePoint, whichWindow, whichControl);
    IF whichControl <> NIL THEN
      BEGIN
        writeln('TrackControl : ');
        whichPart :=
        TrackControl(whichControl, thePoint, pointer(-1));
      END;
    UNTIL whichControl = NIL;

  FlushEvents(MUPMask, 0);
  ( let's have another loop to test "DragControl" (and "FindControl"): )
  writeln('The user should try to drag the control. ');
  writeln('End by clicking outside the control. ');
  REPEAT
    REPEAT
      UNTIL GetNextEvent(MDownMask, theEvent);
      thePoint := theEvent.where;
      whichPart := FindWindow(thePoint, whichWindow);
      SetPort(whichWindow); GlobalToLocal(thePoint);
      writeln('FindControl : ');
      whichPart :=
      FindControl(thePoint, whichWindow, whichControl);
      IF whichControl <> NIL THEN
        WITH whichWindow^ DO
          BEGIN
            writeln('DragControl : ');
            DragControl(whichControl, thePoint, PortRect,
            PortRect, noConstraint);
          END;
        UNTIL whichControl = NIL;
        writeln('DisposeControl : '); DisposeControl(myControl);
        writeln('end');
      1 :
      ( let's remove our fake CDEF : )
      RmveResource(handle(myFakeCDEF));
      IF ResError <> NoErr THEN
        GOTO 2;
      DisposHandle(handle(myFakeCDEF));

      ( error label : )
      2 :
      END.

      (*****
      ( BUILDING THE MDEF CODE RESOURCE )
      (*****
      ( Put this file in the MPOP Project after DAPasLib. )
      ( Don't put MacTraps that would generate unusefull glue for
      the Memory Manager. )
      ( I prefer to declare DisposHandle as inline procedure : see
      below. )
      ( Don't forget to "Use resource file" in "Run options" of menu
      "Project". )
      ( This resource file must contain the MENU and ICN# resources )
      ( that the PopTrap Project needs together with the compiled
      MDEF resource. )

```

```

( "Build and save as..." resource code of type MDEF and ID 128
in file "MPOP code" )
(*****
UNIT MPOP;

INTERFACE
( the name "Main" indicates to LightSpeed Pascal compiler
where the entry point is )
PROCEDURE Main (message : integer; theMenu : MenuHandle; VAR
menuRect : rect; hitPt : point; VAR whichItem : integer);

IMPLEMENTATION
CONST
  mPopUpMsg = 3;
( and not 4 as written in early versions of new MenuMgr )

PROCEDURE CopyMask (srcBits, maskBits, dstBits : BitMap;
srcRect, maskRect, dstRect : Rect);
  INLINE $A817;

PROCEDURE DisposHandle (h : handle);
( to avoid putting the whole Memory Manager glue in our code
resource )
  INLINE $205F, $A023, $31C0, $0220;

(*****
( first some utilities for reading MENU resources : )

FUNCTION GetNextByte (VAR LongAddress : longint) : byte;
  INLINE
    $205F, ( MOVEA.L (A7)+,A0 )
    $2250, ( MOVEA.L (A0),A1 )
    $5290, ( ADDQ.L #1,(A0) )
    $204F, ( MOVEA.L A7,A0 )
    $4218, ( CLR.B (A0)+ )
    $1091, ( MOVE.B (A1),(A0) )

FUNCTION GetNextInteger (VAR LongAddress : longint) :
integer;
  INLINE
    $205F, ( MOVEA.L (A7)+,A0 )
    $2250, ( MOVEA.L (A0),A1 )
    $5490, ( ADDQ.L #2,(A0) )
    $204F, ( MOVEA.L A7,A0 )
    $10D9, ( MOVE.B (A1)+,(A0)+ )
    $1091, ( MOVE.B (A1),(A0) )

FUNCTION GetNextString (VAR LongAddress : longint) :
StringHandle;
( returns NIL if allocation failed )
  INLINE
    $205F, ( MOVEA.L (A7)+,A0 ;A0:=@LongAddress )
    $2250, ( MOVEA.L (A0),A1 ;A1:=LongAddress )
    $7000, ( MOVEQ #0,D0 ;countChars:=0 )
    $1011, ( MOVE.B (A1),D0 ;countChars:=LongAddress^ )
    $2200, ( MOVE.L D0,D1 ;save countChars )
    $5200, ( ADDQ.B #1,D0 ;length:=countChars+1 )
    $D190, ( ADD.L D0,(A0)
    ;FuturLongAddress:=LongAddress+length )
    $A122, ( OSTRAP $A122 ;A0:=NewHandle(D0=length) )
    $4A80, ( TST.L D0 ;if MemError )
    $660C, ( BNE.S *+$000E ;<>0 goto error )
    $2E88, ( MOVE.L A0,(A7) ;GetChaine:=A0 )
    $2050, ( MOVEA.L (A0),A0 ;StringPtr )
    ( loop ;repeat )
    $10D9, ( MOVE.B (A1)+,(A0)+
    ;StringPtr:=LongAddress^ )
    $51C9, $FFFC, ( DBF D1,*-$0002
    ;dec(length); until length<0 )
    $6002, ( BRA.S *+$0004 ;goto bottom )
    ( error )
    $4297, ( CLR.L (A7) ;GetChaine:=NIL )
    ( bottom )

```

```

FUNCTION SkipNextString (VAR LongAddress : longint) : byte;
VAR length : byte;
BEGIN
    length := GetNextByte(LongAddress);
    LongAddress := LongAddress + length;
    SkipNextString := length;
END;

PROCEDURE SkipBytes (VAR LongAddress : longint;
                     byteCount : integer);
BEGIN
    LongAddress := LongAddress + byteCount;
END;

(*****)
PROCEDURE Main;

    FUNCTION GetItemCenter : point;
    ( returns the ItemCenter in local coordinates, relative to
      menuRect )
    ( theMenu is already locked )
    VAR
        LongAddress : longint;
        length : byte;
        i : integer;
        ItemCenter : point;
    BEGIN
        LongAddress := ord(theMenu^) + 14;
        length := SkipNextString(LongAddress);
        i := 0;
        REPEAT
            i := i + 1;
            length := SkipNextString(LongAddress);
            IF length > 0 THEN
                BEGIN
                    IF i = whichItem THEN
                        BEGIN
                            ItemCenter.v := GetNextInteger(LongAddress);
                            ItemCenter.h := GetNextInteger(LongAddress);
                        END
                    ELSE
                        BEGIN
                            SkipBytes(LongAddress, 4);
                        END
                    END
                UNTIL (length <= 0) OR (i = whichItem);
                GetItemCenter := ItemCenter;
            END;
        ELSE ( if length <= 0 : )
            SetPt(ItemCenter, 0, 0);
        UNTIL (length <= 0) OR (i = whichItem);
        GetItemCenter := ItemCenter;
    END;

(*****)
PROCEDURE DoDrawMessage;

    PROCEDURE PinString (theString : Str255;
                        center : point);
    BEGIN
        WITH center DO
            MoveTo(h - StringWidth(theString) DIV 2, v);
            DrawString(theString);
        END;

    PROCEDURE PlotIconDataCopy
    (theIcon : handle; dstSquare : rect);
    VAR
        srcSquare : rect;
        data : bitmap;
        myPort : GrafPtr;
    BEGIN
        IF (theIcon <> NIL) THEN
            BEGIN
                SetRect(srcSquare, -16, -16, 16, 16);
                data.rowBytes := 4;
                data.baseAddr := ptr(theIcon^);

```

```

                data.bounds := srcSquare;
                GetPort(myPort);
                CopyBits(data, myPort^.portbits, srcSquare,
dstSquare, srcCopy, NIL);
            END;
        END;

    VAR
        IconRect : rect; IconName : StringHandle;
        LongAddress : longint; NameLength : byte;
        ItemCenter, TextCenter : point;
        theIcon : handle;
    BEGIN
        LongAddress := ord(theMenu^) + 14;
        NameLength := SkipNextString(LongAddress);
        REPEAT
            IconName := GetNextString(LongAddress);
            NameLength := length(IconName^);
            IF NameLength > 0 THEN
                BEGIN
                    theIcon := GetNamedResource('ICN$', IconName^);
                    ItemCenter.v := GetNextInteger(LongAddress) +
menuRect.top;
                    ItemCenter.h := GetNextInteger(LongAddress) +
menuRect.left;
                    WITH ItemCenter DO
                        BEGIN
                            SetRect(IconRect, h - 16, v - 21, h + 16, v + 11);
                            SetPt(TextCenter, h, v + 20);
                        END;
                        PlotIconDataCopy(theIcon, IconRect);
                        TextFont(geneva);
                        TextSize(9);
                        PinString(IconName^, TextCenter);
                        TextFont(systemFont);
                        TextSize(12);
                    END;
                    DisposHandle(handle(IconName));
                UNTIL NameLength <= 0;
            END;
        ( of DoDrawMessage )

(*****)
PROCEDURE DoChooseMessage;

    FUNCTION GetIconRect : rect;
    ( returns the IconRect in global coordinates )
    VAR
        ItemCenter : point;
        IconRect : rect;
    BEGIN
        ItemCenter := GetItemCenter;
        WITH ItemCenter DO
            BEGIN
                IF (h = 0) AND (v = 0) THEN
                    SetRect(IconRect, 0, 0, 0, 0);
                ELSE
                    SetRect(IconRect, h - 16, v - 21, h + 16, v + 11);
                END;
                WITH menuRect DO
                    OffsetSetRect(IconRect, left, top);
                    GetIconRect := IconRect;
                END;
            ( of GetIconRect )
        END;

    PROCEDURE PlotIconMaskXor
    (theIcon : handle; dstSquare : rect);
    VAR
        srcSquare : rect; mask : bitmap;
        myPort : GrafPtr;
    BEGIN
        IF (theIcon <> NIL) THEN
            BEGIN
                SetRect(srcSquare, -16, -16, 16, 16);
                mask.rowBytes := 4;
                mask.baseAddr := ptr(ord4(theIcon^) + 128);

```

```

        mask.bounds := srcSquare; GetPort(myPort);
        CopyBits(mask, myPort^.portbits, srcSquare,
dstSquare, srcX0r, NIL);
    END;
END;      ( of PlotIconMaskXor )

FUNCTION GetIconName (whichItem : integer) : String-
Handle; ( theMenu is already locked )
VAR
    LongAddress : longint; length : byte;
    i : integer; IconName : StringHandle;
BEGIN
    LongAddress := ord(theMenu^) + 14;
    length := SkipNextString(LongAddress);
    i := 0;
    REPEAT
        i := i + 1;
        IF i = whichItem THEN
            BEGIN
                IconName := GetNextString(LongAddress);
            END
        ELSE
            BEGIN
                length := SkipNextString(LongAddress);
                IF length > 0 THEN
                    SkipBytes(LongAddress, 4)
                ELSE
                    IconName := NIL;
                END;
            END
        UNTIL (length <= 0) OR (i = whichItem);
    GetIconName := IconName;
END;

PROCEDURE InvertIcon
(whichItem : integer; dstSquare : rect);
VAR
    IconName : StringHandle;
    myIcon : handle;
BEGIN
    IconName := GetIconName(whichItem);
    myIcon := GetNamedResource('ICN$', IconName^);
    PlotIconMaskXor(myIcon, dstSquare);
END;

VAR
    itemNumber : integer; NameLength : byte;
    LongAddress : longint; ItemCenter : point;
    ItemRect, OldIconRect, IconRect : rect;
BEGIN      ( DoChooseMessage )
    LongAddress := ord(theMenu^) + 14;
    NameLength := SkipNextString(LongAddress);
    itemNumber := 0;
    REPEAT
        itemNumber := itemNumber + 1;
        NameLength := SkipNextString(LongAddress);
        IF NameLength > 0 THEN
            BEGIN
                ItemCenter.v := GetNextInteger(LongAddress);
                ItemCenter.h := GetNextInteger(LongAddress);
                WITH ItemCenter DO
                    SetRect(ItemRect, h - 25, v - 25, h + 25, v + 25);
                WITH menuRect DO
                    OffSetRect(ItemRect, left, top);
                END;
            END
        UNTIL (NameLength <= 0) OR
            (PtInRect(hitPt, ItemRect));
    IF NameLength <= 0 THEN
        ( hitPt is not in any item )
        BEGIN
            IF whichItem <> 0 THEN
                BEGIN
                    InvertIcon(whichItem, GetIconRect);
                    whichItem := 0;
                END;
            END;
        END;
    END;
END;

```

```

    END
ELSE IF itemNumber <> whichItem THEN
    ( hitPt is in itemRect )
    BEGIN
        IF whichItem <> 0 THEN
            InvertIcon(whichItem, GetIconRect);
        WITH ItemCenter DO
            SetRect(IconRect, h - 16, v - 21, h + 16, v + 11);
        WITH MenuRect DO
            OffSetRect(IconRect, left, top);
            InvertIcon(itemNumber, IconRect);
            whichItem := itemNumber;
        END;
    END;
END;      ( of DoChooseMessage )

(*****)
PROCEDURE DoSizeMessage;
( theMenu is already locked )

PROCEDURE RectAndPt (VAR theRect : rect;
thePoint : point);
BEGIN
    WITH theRect, thePoint DO
        ( we suppose that 0=left<right and 0=top<bottom )
        BEGIN
            IF h > right THEN right := h;
            IF v > bottom THEN bottom := v;
        END;
    END;
END;

VAR
    LongAddress : longint; length : byte;
    ItemCenter : point; Envelope : rect;
BEGIN
    LongAddress := ord(theMenu^) + 14;
    length := SkipNextString(LongAddress);
    SetRect(Envelope, 0, 0, 0, 0);
    REPEAT
        length := SkipNextString(LongAddress);
        IF length > 0 THEN
            BEGIN
                ItemCenter.v := GetNextInteger(LongAddress);
                ItemCenter.h := GetNextInteger(LongAddress);
                RectAndPt(Envelope, ItemCenter);
            END
        UNTIL (length <= 0);
    WITH theMenu^, envelope DO
        BEGIN
            menuWidth := right + 25;
            menuHeight := bottom + 25;
        END;
    END;
END;      ( of DoSizeMessage )

(*****)
PROCEDURE DoPopUpMessage;
( on entry: whichItem(=popUpItem) , )
( hitPt (= center of title icon) )
( theMenu (Locked) )
( on exit : menuRect )
( ThePort is already set to WindowManager Port )
VAR
    ItemCenter, IconCenter : point; dh, dv : integer;
    WMPort : GrafPtr; mBarHeight : ^integer;
BEGIN
    mBarHeight := pointer($BAA);
    WITH theMenu^, hitPt DO
        SetRect(menuRect, h, v, h + menuWidth, v + menuHeight);
        IF whichItem > 0 THEN
            BEGIN
                ItemCenter := GetItemCenter;
                WITH ItemCenter DO
                    SetPt(IconCenter, h, v - 5);
                WITH IconCenter DO
                    IF NOT ((h = 0) AND (v = 0)) THEN

```

```

        OffSetRect(menuRect, -h, -v)
    ELSE
        whichItem := 0;
    END;
    IF whichItem <= 0 THEN
        OffSetRect(menuRect, -25, +25);
        GetPort(WMPort);
        WITH WMPort DO
            BEGIN
                IF menuRect.right + 8 > PortRect.right THEN
                    dh := PortRect.right - menuRect.right - 8
                ELSE IF menuRect.left - 8 < PortRect.left THEN
                    dh := PortRect.left - menuRect.left + 8
                ELSE dh := 0;

                IF menuRect.bottom + 8 > PortRect.bottom
                THEN
                    dv := PortRect.bottom - menuRect.bottom - 8
                ELSE IF
                    menuRect.top - 8 < PortRect.top + mBarHeight*
                THEN
                    dv := PortRect.top + mBarHeight*
                        - menuRect.top + 8
                ELSE dv := 0;
            END;
            OffSetRect(menuRect, dh, dv);
        END; ( of DoPopUpMessage )

    (*****
    BEGIN ( of Main )
        CASE message OF
            mSizeMsg : DoSizeMessage;
            mDrawMsg : DoDrawMessage;
            mChooseMsg : DoChooseMessage;
            mPopUpMsg : DoPopUpMessage;
        END;
    END;

END.
(*****
( BUILDING THE CDEF CODE RESOURCE )
(*****

( Put this file in the CDEF Project after DAPasLib, MacTraps,
ROM851ib and ROM85. )
( Don't forget to "Use resource file" in "Run options" of menu
"Project". )
( This resource file must contain the WIND ,CNTL, MENU, ICN*,
ICON, MDEF )
( resources that the Shell Project needs together with the
compiled CDEF resource. )
( "Build and save as..." resource code of type CDEF and ID 128
in file "CDEF code" )
(*****
UNIT CDEF;
INTERFACE
    USES ROM85;

    FUNCTION Main (varCode : integer;
        theControl : ControlHandle; message : integer;
            param : longint) : longint;

IMPLEMENTATION
CONST
    RestState = 0; SelectState = 1; OpenState = 2;
    SelectOpenState = 3; ThrownAwayState = 4;
    MenuReturnState = 5;

    movableBit = 1; doubleClickableBit = 2;
    trashBit = 3; menuBit = 4;
    varCodeBase = 200;
    { bit-offset of end of varCode in ControlRecord }
    integerLength = 16;
TYPE
    DataHandle = ^DataPointer;

```

```

    DataPointer = ^DataRecord;
    DataRecord = RECORD
        theIcon : handle;
        theMenu : MenuHandle;
    END;

FUNCTION PopUpMenuSelect (menu : MenuHandle;
    top, left, popUpItem : integer) : longint;
INLINE
    $A80B;

PROCEDURE PlotDoubleIcon (theIcon : handle;
    State : integer; dstSquare : rect);
VAR
    srcSquare : rect;
    data, mask, destBitMap, scratchBitMap : bitmap;
    theGrafPort : GrafPtr;
    LightGrayIcon, DarkGrayIcon : handle;
BEGIN
    IF (theIcon <> NIL) THEN
        BEGIN
            SetRect(srcSquare, -16, -16, 16, 16);

            data.rowBytes := 4;
            data.baseAddr := ptr(theIcon^);
            data.bounds := srcSquare;

            mask.rowBytes := 4;
            mask.baseAddr := ptr(ord4(theIcon^) + 128);
            mask.bounds := srcSquare;

            GetPort(theGrafPort);
            destBitMap := theGrafPort^.portbits;
            CASE state OF
                RestState :
                    BEGIN
                        CopyBits(mask, destBitMap, srcSquare, dstSquare, srcBic,
                            NIL);
                        CopyBits(data, destBitMap, srcSquare, dstSquare, srcOr,
                            NIL);
                    END;
                SelectState :
                    BEGIN
                        { old finder : }
                        CopyBits(mask, destBitMap, srcSquare, dstSquare, srcBic,
                            NIL);
                        CopyBits(data, destBitMap, srcSquare, dstSquare, srcOr,
                            NIL);
                        CopyBits(mask, destBitMap, srcSquare, dstSquare, srcXOr,
                            NIL);
                        { new finder would be : }
                        {CopyBits(mask, destBitMap, srcSquare, dstSquare, srcOr,
                            nil);}
                        {CopyBits(data, destBitMap, srcSquare, dstSquare, srcBic,
                            nil);}
                    END;
                OpenState :
                    BEGIN
                        WITH scratchBitMap DO
                            BEGIN
                                LightGrayIcon := GetIcon(128);
                                BaseAddr := LightGrayIcon^;
                                bounds := srcSquare;
                                Rowbytes := 4;
                            END;
                            CopyBits(mask, destBitMap, srcSquare,
                                dstSquare, srcBic, NIL);
                            CopyMask(scratchBitMap, mask, destBitMap,
                                srcSquare, srcSquare, dstSquare);
                        END;
                        SelectOpenState :
                            BEGIN
                                WITH scratchBitMap DO
                                    BEGIN

```

```

        DarkGrayIcon := GetIcon(129);
        BaseAddr := DarkGrayIcon^;
        bounds := srcSquare;
        RowBytes := 4;
        END;
        CopyBits(mask, destBitMap, srcSquare,
dstSquare, srcBic, NIL);
        CopyMask(scratchBitMap, mask, destBitMap,
srcSquare, srcSquare, dstSquare);
        END;
    OTHERWISE
        END;
    END
END;      ( of PlotDoubleClick )

( ***** )
FUNCTION distance (startPt, endPt : point) : integer;
BEGIN
    distance := abs(startPt.h - endPt.h) + abs(startPt.v -
endPt.v);
END;

FUNCTION InsideIcon (myPoint : point;
                    IconCenter : point;
                    myIcon : handle) : boolean;

VAR
    bitOffset : longint;
    scratchMap, dataMap, maskMap, sensitiveMap : bitmap;
    square : rect;
    x, y : integer;
LABEL
    1;
BEGIN
    HLock(myIcon);
    SetRect(square, 0, 0, 32, 32);
    WITH scratchMap DO
        BEGIN
            bounds := square;
            BaseAddr := NewPtr(128);
            IF MemError <> NoErr THEN
                GOTO 1;
            RowBytes := 4;
        END;
    WITH sensitiveMap DO
        BEGIN
            bounds := square;
            BaseAddr := NewPtr(128);
            IF MemError <> NoErr THEN
                GOTO 1;
            RowBytes := 4;
        END;
    WITH dataMap DO
        BEGIN
            bounds := square;
            BaseAddr := myIcon^;
            RowBytes := 4;
        END;
    WITH maskMap DO
        BEGIN
            bounds := square;
            BaseAddr := Ptr(Ord4(myIcon^) + 128);
            RowBytes := 4;
        END;
        CopyBits(maskMap, scratchMap, square, square, srcCopy,
NIL);
        CopyBits(dataMap, scratchMap, square, square, srcOr, NIL);
        CalcMask(scratchMap.baseAddr, sensitiveMap.baseAddr, 4, 4,
32, 2);
        x := myPoint.h - IconCenter.h + 16;
        y := myPoint.v - IconCenter.v + 16;
        IF NOT ((x IN [0..31]) AND (y IN [0..31])) THEN
            InsideIcon := false
        ELSE
            BEGIN

```

```

                bitOffset := x + 32 * y;
                InsideIcon := BitTst(sensitiveMap.baseAddr, bitOff-
set);
            END;
            HUnlock(myIcon);
            DisposPtr(scratchMap.baseAddr);
            DisposPtr(sensitiveMap.baseAddr);
        1 :
            IF MemError <> NoErr THEN
                InsideIcon := false;
            END;

FUNCTION DoubleClick (theControl : ControlHandle;
startPt : point; startTime : longint; VarCode : integer;
bounds : rect; VAR IconCenter : point) : boolean;
VAR
    mouse : point; t : longint; d : integer;
    theEvent : EventRecord; DoubleClicked : boolean;

PROCEDURE DragSquare (startPt : point;
                    VAR IconCenter : point);

VAR
    oldFrame, frame, bounds : rect;
    delta, mouse : point; theGrafPort : GrafPtr;
    grayPattern : pattern; theTrash : ControlHandle;

PROCEDURE HighLightTrash (mouse : point);
VAR
    where : integer; IconCenter : point;
    OverFlownControl : ControlHandle;
BEGIN
    where := FindControl(mouse, FrontWindow, Over-
FlownControl);
    HLock(GetResource('CDEF', 128));
    FrameRect(oldFrame);
    IF theTrash <> OverFlownControl THEN
        ( the control the mouse overflies is no more "theTrash" )
        BEGIN
            IF theTrash <> NIL THEN
                ( the mouse has ended overflying a trash )
                BEGIN
                    SetCtlValue(theTrash, GetCtlValue(theTrash) - 1);
                    HLock(theTrash^.ContrlDefProc);
                END;
                IF (OverFlownControl <> NIL) THEN
                    IF BitTst(pointer(OverFlownControl^),
varCodeBase - trashBit) AND
                    (theControl <> OverFlownControl)
                    THEN
                        ( the mouse begins overflying a trash )
                        BEGIN
                            SetCtlValue(OverFlownControl,
                                GetCtlValue(OverFlownControl) + 1);
                            HLock(OverFlownControl^.ContrlDefProc);
                            theTrash := OverFlownControl;
                        END
                    ELSE
                        ( the mouse overflies something else than a trash )
                        theTrash := NIL
                    ELSE
                        ( the mouse doesn't overfly anything )
                        theTrash := NIL;
                    END;
                END;
            BEGIN
                ( DragSquare )
                theTrash := NIL;
                StuffHex(@grayPattern, '55AA55AA55AA55AA');
                GetPort(theGrafPort);
                bounds := theGrafPort^.PortRect;
                InSetRect(bounds, 16, 16);
                delta := IconCenter;
                SubPt(startPt, delta);
                PenMode(PatXor);

```

```

WITH IconCenter DO
  SetRect(oldFrame, h - 16, v - 16, h + 16, v + 16);
  PenPat(grayPattern);
  FrameRect(oldFrame);
( instead of the surrounding square )
( we could also drag the icon's data or mask frame )
REPEAT
  GetMouse(mouse); IconCenter := mouse;
  AddPt(delta, IconCenter);
  WITH IconCenter, bounds DO
    BEGIN
      IF h < left THEN h := left;
      IF h > right - 1 THEN h := right - 1;
      IF v < top THEN v := top;
      IF v > bottom - 1 THEN v := bottom - 1;
    END;
  WITH IconCenter DO
    SetRect(frame, h - 16, v - 16, h + 16, v + 16);
    IF NOT EqualRect(oldFrame, frame) THEN
      BEGIN
        HighlightTrash(mouse);
        FrameRect(frame); oldFrame := frame;
      END;
    UNTIL NOT WaitMouseUp;
    FrameRect(frame); PenNormal;
  END;
END;

BEGIN ( DoubleClick )
  DoubleClicked := false;
  BEGIN
    ( if doubleClickable or movable : )
    IF (BitTst(@varCode,
      integerLength - doubleClickableBit)) OR
      (BitTst(@varCode, integerLength - movableBit)) THEN
      REPEAT
        GetMouse(mouse);
        d := distance(startPt, mouse);
        UNTIL (NOT WaitMouseUp OR (d > 3));
        IF (d > 3) AND BitTst(@varCode, integerLength -
movableBit) THEN
          DragSquare(startPt, IconCenter)
        ELSE IF BitTst(@varCode,
          integerLength - doubleClickableBit) THEN
          REPEAT
            GetMouse(mouse);
            d := distance(startPt, mouse);
            t := TickCount - startTime;
            IF GetNextEvent(MDownMask, theEvent) THEN
              DoubleClicked := true;
              UNTIL DoubleClicked OR (d > 3) OR (t > GetDbtTime);
              DoubleClick := DoubleClicked;
            END;
          END;
        END;
      END;

( ***** )
FUNCTION Main;
  VAR
    ( color under the title : )
    whitePattern : pattern;

  PROCEDURE DoDrawCnt1;
  VAR
    IconCenter, TextCenter : point;
    State, theLength, theHalfLength : integer;
    TextFrame, IconFrame : rect;
    myDataHandle : DataHandle;
  BEGIN
    State := GetCtlValue(theControl);
    ( MenuReturnState is drawn like RestState, ThrownAwayState is
not re-drawn : )
    IF State = MenuReturnState THEN
      State := RestState;
    IF ((State IN (RestState, SelectOpenState)) AND
(theControl^.Contr1Vis <> 0)) THEN

```

```

BEGIN
  HLock(handle(theControl));
  WITH theControl^^ DO
    BEGIN
      TextFont(geneva); TextFace(1);
      TextMode(SrcOr); TextSize(9);
      theLength := StringWidth(contr1Title);
      IF theLength < 32 THEN
        theHalfLength := 16
      ELSE
        theHalfLength := theLength DIV 2;
      WITH Contr1Rect DO
        SetPt(IconCenter, (right + left) DIV 2,
          (bottom - 12 + top) DIV 2);
      ( recalculate the rect surrounding the whole control : )
      WITH IconCenter, Contr1Rect DO
        BEGIN
          left := h - theHalfLength; top := v - 16;
          right := h + theHalfLength; bottom := v + 16 + 12;
        END;

      WITH IconCenter, IconFrame DO
        BEGIN
          left := h - 16; top := v - 16;
          right := h + 16; bottom := v + 16;
        END;

      ( draw the icon-control's title : )
      WITH IconCenter DO
        SetPt(TextCenter, h, v + 26);
      WITH TextCenter DO
        SetRect(TextFrame, h - theLength DIV 2,
          v - 10, h + theLength DIV 2, v + 2);
        StuffHex(@whitePattern, '0000000000000000');
        FillRect(TextFrame, whitePattern);
      WITH TextCenter DO
        MoveTo(h - theLength DIV 2, v);
        DrawString(contr1Title);
      ( draw the icon : )
      myDataHandle := DataHandle(Contr1Data);
      HLock(myDataHandle^.theIcon);
      PlotDoubleIcon
        (myDataHandle^.theIcon, State, IconFrame);
      HUnlock(myDataHandle^.theIcon);
    END;
  HUnlock(handle(theControl));
END;
Main := 0;
END;

PROCEDURE DoTestCnt1;
VAR
  IconCenter, mouse : point;
  myDataHandle : DataHandle;
BEGIN
  HLock(handle(theControl));
  WITH theControl^^ DO
    BEGIN
      SetPt(mouse, LoWord(param), HiWord(param));
      IF PtInRect(mouse, Contr1Rect) THEN
        BEGIN
          WITH Contr1Rect DO
            SetPt(IconCenter, (right + left) DIV 2,
              (bottom - 12 + top) DIV 2);
            myDataHandle := DataHandle(Contr1Data);
            Main := ord4(InsideIcon(mouse, IconCenter,
              myDataHandle^.theIcon));
          END
        ELSE
          main := 0;
        END;
      END;
    HUnlock(handle(theControl));
  END;

```

```

PROCEDURE DoCalcCRgns;
CONST
    Lo3Bytes = $00FFFFFF;
VAR
    IconFrame, TextFrame : rect; theTitle : Str255;
    theLength, theHalfLength, halfWay : integer;
BEGIN
    GetCtlValue(theControl, theTitle);
    theLength := StringWidth(theTitle);
    theHalfLength := theLength DIV 2;
    param := BitAnd(param, Lo3Bytes);
    IconFrame := theControl^.Contr1Rect;
    WITH IconFrame DO
        BEGIN
            bottom := bottom - 12; halfWay := (right + left) DIV 2;
            left := halfWay - 16; right := halfWay + 16;
            SetRect(TextFrame, halfWay - theHalfLength,
                bottom, halfWay + theHalfLength, bottom + 12);
        END;
        OpenRgn;
        FrameRect(IconFrame); FrameRect(TextFrame);
        CloseRgn(RgnHandle(param));
        Main := 0;
END;

PROCEDURE DeselectExcept
    (theControl : ControlHandle);

VAR
    myWindowPeek : WindowPeek;
    aControl : ControlHandle;
BEGIN
    myWindowPeek :=
        WindowPeek(theControl^.Contr1Owner);
    aControl := myWindowPeek^.ControlList;
    WHILE aControl <> NIL DO
        BEGIN
            IF (aControl <> theControl) THEN
                BEGIN
                    IF (GetCtlValue(aControl) = 1) THEN
                        BEGIN
                            SetCtlValue(aControl, 0);
                            HLock(aControl^.Contr1DefProc);
                        END
                    ELSE IF
                        (GetCtlValue(aControl) = 3) THEN
                            BEGIN
                                SetCtlValue(aControl, 2);
                                HLock(aControl^.Contr1DefProc);
                            END
                        END;
                        aControl := aControl^.nextControl;
                    END;
                END;
        END;

PROCEDURE DoAutoTrack;
VAR
    SavedClip, UpDateRegion : RgnHandle;
    PopUpMenuHdl : MenuHandle;
    (MDEFPtr : Ptr; ( for debugging only )
    theTitle : Str255;
    choosenItem, dummy : longint;
    halfWay, theHalfLength, where : integer;
    oldCenter, IconCenter, mouse, MenuTitleCenter : point;
    theGrafPort : GrafPtr;
    theTrash : ControlHandle;
    myDataHandle : DataHandle;
    isAMenu, isDoubleClickable, isMovable : boolean;
BEGIN
    isAMenu := BitTst(@varCode, integerLength - MenuBit);
    isDoubleClickable := BitTst(@varCode, integerLength -
        DoubleClickableBit);
    isMovable := BitTst(@varCode, integerLength - MovableBit);
    IF isAMenu OR isDoubleClickable OR isMovable THEN
        BEGIN

```

```

            IF GetCtlValue(theControl) = OpenState THEN
                SetCtlValue(theControl, SelectOpenState)
            ELSE IF GetCtlValue(theControl) = RestState
            THEN SetCtlValue(theControl, SelectState);
                HLock(theControl^.Contr1DefProc);
            END;
            DeselectExcept(theControl);
            GetMouse(mouse);
            WITH theControl^.Contr1Rect DO
                SetPt(IconCenter, (right + left) DIV 2,
                    (bottom - 12 + top) DIV 2);
                oldCenter := IconCenter; GetPort(theGrafPort);

        ( 1° : DOUBLE-CLICK )
            IF DoubleClick(theControl, mouse, TickCount, varCode,
                theGrafPort^.PortRect, IconCenter) THEN
                BEGIN
                    SetCtlValue(theControl, SelectOpenState);
                    HLock(theControl^.Contr1DefProc);
                END

        ( 2° : NO DRAGGING )
            ELSE IF EqualPt(oldCenter, IconCenter) THEN
                BEGIN

        ( 2.1 : POPUPMENU )
            IF isAMenu THEN
                BEGIN
                    myDataHandle := DataHandle(theControl^.Contr1Data);
                    PopUpMenuHdl := myDataHandle^.theMenu;

                    WITH theControl^.Contr1Rect, MenuTitleCenter DO
                        BEGIN
                            h := (left + right) DIV 2;
                            v := (top + bottom) DIV 2;
                        END;
                        LocalToGlobal(MenuTitleCenter);
                        WITH MenuTitleCenter DO
                            choosenItem := PopUpMenuSelect(PopUpMenuHdl, h, v, 0);
                            ( re-draw the control as in RestState : )
                            SetCtlValue(theControl, MenuReturnState);
                            HLock(theControl^.Contr1DefProc);
                            SetCRefCon(theControl, choosenItem);
                        END

        ( 2.2 : SIMPLE SELECTION OF A DOUBLE-CLICKABLE CONTROL )
            ( the Control is already highlighted in the "SelectState" )
            END

        ( 3° : DRAGGING )
            ELSE
                BEGIN
                    GetMouse(mouse);
                    where := FindControl(mouse, FrontWindow, theTrash);
                    HLock(GetResource('CDEF', 128));
                    IF (theTrash <> NIL) THEN

        ( 3.1 : THROWING THE CONTROL AWAY IN A TRASH )
                        IF BitTst(pointer(theTrash),
                            varCodeBase - trashBit)
                            AND (theTrash <> theControl)
                            THEN
                                BEGIN
                                    ( return "theTrash" in CRefCon )
                                    ( without re-drawing it )
                                    SetCRefCon(theControl, ord(theTrash));
                                    SetCtlValue(theControl, ThrownAwayState);
                                    HLock(theControl^.Contr1DefProc);
                                END;
                            END;

        ( 3.2 : MOVING )
                        IF (GetCtlValue(theControl) <>
                            ThrownAwayState) THEN
                            BEGIN
                                WITH theControl^.Contr1Rect DO

```

```

        theHalfLength := (right - left) DIV 2;
        ( move the control without showing it : )
        HideControl(theControl);
        HLock(theControl^.ContrlDefProc);
        WITH IconCenter DO
            MoveControl(theControl,
                h - theHalfLength, v - 16);
        HLock(theControl^.ContrlDefProc);
        theControl^.ContrlValue := RestState;

( the UpDate mechanism will do the re-drawing in such a way it
lets the previously hidden controls appear : )

        SavedClip := NewRgn;
        GetClip(SavedClip);
        SetEmptyRgn(theGrafPort^.ClipRgn);
        ShowControl(theControl);
        HLock(theControl^.ContrlDefProc);
        SetClip(SavedClip);
        ( re-use an initialised region for another purpose : )
        UpDateRegion := SavedClip;
        ( send the CalcCRgns message to calculate UpDateRegion : )
        dummy := Main(0, theControl,
            calcCRgns, ord4(UpDateRegion));
        EraseRgn(UpDateRegion);
        InValRgn(UpDateRegion);
        DisposeRgn(UpDateRegion);
    END;
END;
Main := 0;
END;

PROCEDURE DoInitCntl;
VAR
    myDataHandle : DataHandle;
    theTitle : Str255;
BEGIN
    GetCTitle(theControl, theTitle);
    myDataHandle :=
DataHandle(NewHandle(sizeof(DataRecord)));
    HLock(handle(myDataHandle));
    WITH myDataHandle DO
        BEGIN
            theIcon := GetNamedResource('ICN#', theTitle);
            ( Initialisation should have called "GetMenu" : )
            theMenu := MenuHandle(GetNamedResource('MENU',
theTitle));
        END;
        HUnlock(handle(myDataHandle));
        WITH theControl DO
            BEGIN
                ContrlAction := pointer(-1);
                ContrlData := handle(myDataHandle);
            END;
        END;
    END;

PROCEDURE DoDispCntl;
BEGIN
    DisposHandle(theControl^.ContrlData);
END;

BEGIN ( Main procedure )
CASE message OF
    drawCntl : DoDrawCntl;
    testCntl : DoTestCntl;
    calcCRgns : DoCalcCRgns;
    initCntl : DoInitCntl;
    dispCntl : DoDispCntl;
    dragCntl : ( for a smoother interface )
        BEGIN
            DoAutoTrack;
            Main := 1; ( to tell the Control Manager not to
use the standard method )
        END;

```

```

        autoTrack : DoAutoTrack;
    OTHERWISE ( dragCntl, posCntl, thumbCntl )
        main := 0;
    END;
END;

(*****
( SEE HOW SIMPLE IT IS FOR THE END-PROGRAMMER )
( TO CREATE AND USE FINDER-LIKE ICONS! )
(*****
( Put this file in the Shell Project after DAPasLib, MacTraps,
ROM85Lib and ROM85, put also CDEF text if you plan to improve
and debug it. Don't forget to "Use resource file" "CDEF Code"
in "Run options" of menu "Project". This resource file must
contain the WIND ,CNTL, MENU, ICN#, ICON, MDEF and CDEF
resource. )
(*****
PROGRAM shell;
USES ROM85;
    ( , CDEFIcons; ( only for debugging purposes )
CONST
    RestState = 0; SelectState = 1; OpenState = 2;
    SelectOpenState = 3; ThrownAwayState = 4;
    MenuReturnState = 5;
TYPE
    CDEFcodeHdl = ^CDEFcodePtr;
    CDEFcodePtr = ^CDEFcodeRecord;
    CDEFcodeRecord = RECORD
        jump : integer;
        address : ProcPtr;
    END;
VAR
    whichWindow, myWindow : WindowPtr;
    whichControl, theTrash : ControlHandle;
    myCDEF : CDEFcodeHdl; theEvent : EventRecord;
    theControl : ControlHandle; mouse : point;
    FakeRect : rect; done : boolean; i : integer;

PROCEDURE ProcessMenu (where : longint);
VAR
    MenuNb, ItemNb : integer; dummy : longint;
    i : integer;
BEGIN
    MenuNb := HiWord(where);
    ItemNb := LoWord(where);
    CASE MenuNb OF
        1 :
            FOR i := 1 TO ItemNb DO
                BEGIN
                    SysBeep(5);
                    Delay(10, dummy);
                END;
            OTHERWISE
                ;
            END;
    END;
END;

PROCEDURE ProcessEvent;
VAR
    myWindowPeek : WindowPeek;
    dummy : integer;
BEGIN
    CASE theEvent.what OF
        UpDateEvt :
            BEGIN
                whichWindow := WindowPtr(theEvent.message);
                beginUpDate(whichWindow);
                IF whichWindow = myWindow THEN
                    BEGIN
                        EraseRgn(myWindow^.VisRgn);
                        UpdtControl(myWindow, myWindow^.VisRgn);
                    END;
                endUpDate(whichWindow);
            END;

```



```

END;
MouseDown :
BEGIN
CASE FindWindow(theEvent.where, whichWindow) OF
inContent :
BEGIN
mouse := theEvent.where;
GlobalToLocal(mouse);
IF FindControl(mouse, whichWindow, whichControl) <>
Ø THEN
BEGIN
( choose DragControl or TrackControl as you prefer: )
( dummy:=TrackControl(theControl,mouse,pointer(-1)) )
SetRect(FakeRect, Ø, Ø, Ø, Ø);
DragControl(whichControl, mouse, FakeRect, Fake-
eRect, noConstraint);
IF GetCtlValue(whichControl) = ThrownAwayState
THEN
BEGIN
theTrash := ControlHandle(GetCRefCon(whichControl));
SetCtlValue(theTrash, GetCtlValue(theTrash) - 1);
DisposeControl(whichControl);
END
ELSE IF GetCtlValue(whichControl) = MenuReturnState
THEN
BEGIN
ProcessMenu(GetCRefCon(whichControl));
( the Control has already been re-drawn in RestState, )
( so we don't need to re-redraw it again.)
whichControl^.CtrlValue := Ø;
END
ELSE IF GetCtlValue(whichControl) = SelectOpenState
THEN
BEGIN
dummy := NoteAlert(128, NIL);
SetCtlValue(whichControl, RestState);
END
ELSE
;
END;
END;
inGoAway :

```

```

IF TrackGoAway(whichWindow, theEvent.where)
THEN done := true;
inDrag :
DragWindow(whichWindow, theEvent.where,
ScreenBits.bounds);
OTHERWISE
END;
END;
OTHERWISE
END;
END;

BEGIN
( create a intermediate CDEF resource : )
(myCDEF := CDEFcodeHdl(NewHandle(sizeof(CDEFcodeRecord)));
( for debugging only )
(myCDEF^.jump := $4EF9; ( for debugging only )
(myCDEF^.address := @main; ( for debugging only )

(AddResource(handle(myCDEF), 'CDEF', 128, ''); ( for debugging
only )

myWindow := GetNewWindow(128, NIL, pointer(-1));
SetPort(myWindow);
(BackPat(gray); ( or whatever background pattern you wish )
InsertMenu(GetMenu(128), -1);
FOR i := 128 TO 133 DO
theControl := GetNewControl(i, myWindow);
done := false;
SetCursor(arrow);
FlushEvents(EveryEvent, Ø);
ShowWindow(myWindow);
REPEAT
IF GetNextEvent(EveryEvent, theEvent) THEN
ProcessEvent;
UNTIL done;
DisposeWindow(myWindow);
(RmveResource(handle(myCDEF)); ( for debugging only )
(DisposeHandle(handle(myCDEF)); ( for debugging only )
END.

```



Advanced Mac'ing

Security Patrol for Viruses



Steven Seaquist
Washington, DC
Volume 5, Number 2

Politics of Virus Protection

"SecurityPatrol" is a program I wrote to detect and optionally remove the "nVIR" and "Scores" viruses (all Macintosh viruses known at the time I wrote it), plus several other conditions I would find suspicious and would want to be warned about. This is version 1.1. It's been tested against live strains of nVIR and Scores. It contains a code database of "fingerprints" to identify known viruses and detect alterations of a system's "good" (trusted) resources by future viruses. *[The source code disk for this month contains the complete library of system resource fingerprints for Security Patrol. In the interest of space, only a few example fingerprints are printed in the magazine. -Ed]*

It wasn't coded so much to be "user-friendly"; my main goal was to make it "security-stubborn": It's deliberately not easy to alter its code database of resources. You have to code them in manually and recompile. To put it another way, the easier it is for a user to alter the database, the easier it is for a virus to alter the database and masquerade as something innocuous.

On the other hand, there are advantages to the fact that it's being published in source code format. It allows you to re-code it to deal with future viruses and/or to make it deliberately inconsistent with the published version. (A predictable defense is generally a weakness to attack.) Another significant advantage is that it inherently discloses its behavior. Sunlight is a strong disinfectant.

[While Apple Computer cannot and does not endorse anything in this article, it's publication in MacTutor was of sufficient concern that Scott Boyd, a contributing editor of MacTutor and a member of Apple's internal virus committee, asked and was given permission to review the article prior to publication. He indicated Apple's concern that an article of this sort that points out the system weaknesses can also give more ammunition to fuel future virus makers. However, since three books have recently been published on how to create a computer virus, we feel the cat is out of the bag anyway so the protection this technology will allow developers is more vital than ever. Accordingly, we are publishing the article essentially unchanged from the author's version, except for a couple of places where Steven reveals "chinks in the armor" of Apple's system software. Scott did allow that many of Steven's techniques are being used in internal security programs at Apple so we feel this approach has merit. We invite comment on the problem that Scott and others at Apple must wrestle with: namely how much information should the public be given on the internal workings of Apple's system software and it's potential vulnerability to viral attack. -Ed]

User Interface

The program sends feedback to the screen and to a report file, but first the user has to select the report filename with a SFPut-

File (Save As...) dialog. If AppleTalk is active, pressing Cancel will cause it to Quit; if it's not active, pressing Cancel will cause it to stream the text to a direct-connect ImageWriter on the printer port. (There's a bug in TML II that causes WriteLn output to the ImageWriter not to work, but it'll be fixed soon. In the meantime, the user can send the report to a file or press Cancel to avoid creating a report file.) After selecting the report file, the user is presented with the Main Dialog.

Main Dialog's "Scope of Work" Buttons

The user may want to patrol only a few files, or all files in a folder, or all files on several disks bought in a store. Maybe the user doesn't have any particular files in mind, but wants to see whether or not any file on a partitioned hard disk is infected. The program cannot know in advance the scope of work to be done, so it asks the user in its Main Dialog.

Directories: The program puts up an SFGGetFile (Open...) dialog to allow the user to select a file. The program patrols all files under the directory that contains the selected file. Then, under HFS, it recursively patrols all subdirectories. Then it returns to the SFGGetFile dialog. This process repeats until the user presses the Cancel button.

Directory: Same as Directories, except that it doesn't patrol subdirectories.

Everything: The program loops thru all currently mounted volumes and patrols Directories (plural) on each one starting from the root directory.

Files: The program puts up an SFGGetFile dialog to allow the user to select a file, patrols only that file and returns to the SFGGetFile dialog. This process repeats until the user presses the Cancel button.

Quit: The program prints file and resource totals across all patrols before returning to the Finder.

User Interface: Options

The Main Dialog also has options the user (usually to get the user's attention) and for the programmer (usually to help ease the non-trivial programming burden). Both kinds are controlled by check boxes.

Await Keypress: When errors are reported, the program pauses until the user presses a key.

Beep: The program beeps at you when suspicious conditions occur. Complaints get 1 beep, errors get 2 beeps, viral infections get 3 beeps and aborts get 4 beeps. (Most aborts are caused by programming errors, but they could also signify that the program is infected.)

Fingerprint: The program lists the fingerprints of all executable resources it knows about, except CODEs.

Fingerprint CODEs: The program lists the fingerprints of CODE resources too, which generates a lot of output.

Long Listing: The program lists all filenames to its report file, not just those that encountered errors.

Remove Viruses: This is a last resort option for users to recover applications whose backups and masters are lost or unreadable. There are 3 major reasons why you might want to direct users not to use it: concern as to your legal liability if it doesn't remove the virus properly (or damages uninfected applications), a desire to encourage the safer method (restoring from backups) so as to limit future headaches, and the possibility of using the infected disk as evidence in a criminal or civil prosecution (hard to do if you removed the virus as soon as you saw it).

Trace: The program traces its flow. You may want to turn on Await Keypress to step thru it slowly.

Program Design

SecurityPatrol was dually developed under MDS-compatible TML Pascal I, version 2.5 ("TML 2.5") and MPW-compatible TML Pascal II, version 1.0.2 ("TML II"). Both versions are on the source code disk, with TML 2.5 filenames ending in ".Pas" and TMLII filenames ending in ".p". The TML II files are the ones that appear in this article.

Major differences: The TML 2.5 version is roughly 50K and 3% faster, can output to a direct-connect ImageWriter on the printer port and types the report file as an MDS Edit document. The TML II version is roughly 60K, is much easier to develop in, has a larger text I/O window on larger monitors and types the report file as an MPW document.

The program is broken up into 4 major modules: a main program called SecurityPatrol and 3 UNITs called Globals, MainDlog and Patrol. The TML 2.5 versions of these 4 are 99% source code compatible; they differ only in the USES statement. In addition, the Globals UNIT of both versions include the "Fingerprint.ipas" file by means of the {\$I} directive; that is, because it's 100% source code compatible, it's shared. The CodeSizeLimits UNIT is incompatible; there are different versions for TML 2.5 and TML II. Finally, there are BitProcs and PasLibIntf, which are used only in the TML 2.5 version to maintain source code compatibility with TML II.

The files were named such that, under TML 2.5, dependency relationships are properly observed if you simply compile them in alphabetical order. Under TML II, you can let the TML Project Manager and MPW Make facility handle that for you. (I edit the SecurityPatrol.Proj file to recompile Globals.p if Fingerprint.ipas has changed, and to give the Fingerprint and Globals CODE resources the preload and locked attributes.)

Within each module, the procedures are arranged alphabetically for easy lookup. Since they aren't necessarily called in alphabetical order, some routines have to be declared FORWARD.

As far as conversion to other Pascals is concerned, the following may help: The OUTPUT in the PROGRAM header is a TML 2.5 convention to tell the compiler that the program will use WriteLn, etc. TML also defines a Text file variable called OUTPUT that's implicitly used by all screen output WriteLn's, etc, and PasLibIntf procedures. INC and DEC are built-in

functions to increment and decrement a variable by one. (Both are much faster than an assignment such as $x := x + 1$ because they generate the ADDQ and SUBQ instructions.) That's all I can think of for now.

Program Flow

The MainDlog UNIT generates and maintains the Main Dialog display, keyboard equivalents and option check boxes internally. It returns to the main program when a Scope of Work button is pushed.

The SecurityPatrol main program manages program initialization, termination, text I/O and high-level interface between MainDlog and Patrol: Depending on the Scope of Work button pushed, it passes off to the proper routine of Patrol.

The Patrol UNIT contains service routines to scan volumes and/or directories. (It's been programmed to work under both MFS and HFS, but it hasn't been tested under the 64K ROMs.) Its heart and soul is the recursive procedure PatrolDir, which, at the appropriate times, calls 5 routines in Globals: DirectoryBegins, DirectoryEnds, PatrolBegins, PatrolEnds and ProcessFile. Another routine in Patrol, called PatrolFiles (which patrols files manually, a file at a time), also calls those same routines in Globals.

Globals is the largest UNIT and contains all the basic service routines. The first 4 routines just mentioned are currently given only trivial feedback duties at present. The 5th one (ProcessFile) is where all the action starts. If the file being processed contains CODE resources, ProcessFile calls ProcessCodes, which calls LookForKnownViruses and then looks for anomalies in the CODE 0 jump table. Then, for every executable resource type it knows about, ProcessFile calls ProcessRsrcs with the address of a routine to process that resource type. LookForKnownViruses and ProcessRsrcs call routines in Fingerprint to do resource identification. If the Fingerprint routines say that a resource is infected, LookForKnownViruses calls a "Disinfected" routine that knows how to restore the CODE 0 jump table before removing the infected resource(s); ProcessRsrcs calls RemovedRsrc directly to remove it.

Fingerprint is where the "code database" of good (trusted) and bad resource identification resides. A resource's fingerprint is a set of any tests you desire. In this article, 3 tests are used as the fingerprint; on the MacTutor source code disk(s), 7 tests. As new viruses are created, Fingerprint is where you would insert new code to detect them.

Commentary on "CodeSizeLimits.p"

CodeSizeLimits is used by PreprocessSelf in the main program, below. The idea behind it is to prevent a virus from creating a new CODE resource or imbedding itself in one of the existing CODE resources.

Since TML II uses the standard MPW Link tool, which creates 9 CODE resources, the gSizeLimit array is 0..8 and gMaxCode is 8. The size limits for CODEs 1 thru 3 contain a little room for expansion, but not enough for even the tiny nVIR virus to fit into. When you have the program the way you want it, you should set these values to exactly the same size as the CODE resources, so there won't be any room for a virus to sneak into. I have never seen CODEs 4 thru 8 be any size other than exactly

those shown, so I feel comfortable giving them no room for expansion. Unless you add, delete or move subroutines, gJTSIZE will always equal 1240, which represents 155 entries in the jump table. Under TML II, the main program's main procedure cannot be referenced as if it were an external subroutine, because MPW Link doesn't give us a symbolic name for it that we can reference. Tom Leonard suggested defining a subroutine just prior to the main program's main procedure and adding a value to skip over its contents. The value turned out to be \$1A.

Commentary on "Fingerprint.ipas"

FgPr: Fingerprinting begins with a call to InitFgPr and then proceeds by (possibly multiple) calls to FgPr. This allows a data fork to be fingerprinted even when it's larger than the I/O buffer. Calling TML's built-in INC procedure twice is faster than the generic Pascal "sWordPtr := sWordPtr + 2;", but not quite as fast as the new, equivalent TML II syntax: "INC(LONGINT(sWordPtr),2);".

As mentioned above, a resource's fingerprint is a set of tests, any tests you desire. (The only significant restriction is that the test value is a LONGINT.) To fit within MacTutor column width restrictions, the version printed in this article uses only size, checksum, and checkxor. The version on the source code disk(s) uses those 3, plus the number of odd, negative, and positive words, and an alternating checksum/checkdiff. You shouldn't use only the 3 tests in this article, because it's possible to reverse-engineer an evil resource to have the same size, checksum and checkxor as a good one. The 7 tests used on the source code disk(s) make it a virtual certainty that, if two resources have the same fingerprint, they are bit-for-bit identical.

If you want to invest your energies into modifying this program, it wouldn't be very fruitful to devise exotic new fingerprint tests. If you did manage to implement CRC-16, for example, and someone discovered a new virus, how would you find out what its CRC-16 value is?? If no one else has your test, you would need an actual copy of the virus to find out! (Scary!) It's better that there should be just a few common tests, not too computationally intense, the union of which approaches certainty. It would be more fruitful if people devoted their efforts to searching more hiding places. In terms of the "security patrol" analogy, what we need are more guards on patrol, searching in more crevices, not more brands of flashlights.

Good and Evil: Any resource or data fork whose "known" flag is not set to TRUE will be reported as unknown. The Good and Evil routines set this flag. Calls to these routines form the "code database", which presents a tabular appearance in the routines that follow.

Don't be misled by the name "Good". I chose that as its name because it's a natural antonym of Evil and also 4 letters long. It obviously can't know whether or not the original code from the software manufacturer contains malicious code, so it's not known to be "good" in that sense of the term. In the context of what we're doing, it actually means "trusted".

To add the fingerprints of new trusted resources, just run the program against clean disks (masters) with the report being saved to a text file. After quitting the program, open the report file and the Fingerprint.ipas file. The MPW editor allows you to select

everything within a parenthesis pair by double-clicking on the left or right parenthesis of the pair. Using that technique, it's very easy to cut and paste back and forth previously unknown resource fingerprints from the report file to new "IF Good(...) THEN EXIT;" statements in the Fingerprint.ipas file.

SecurityPatrol runs noticeably faster if the fingerprint tests are done in reverse likelihood order. As far as which resources you mark as trusted and the order you test them is concerned, you have to consider what System levels the program will be run under, not just the level you're using for development. On the source code disk(s), I have included major resource fingerprints for Systems 4.2, 4.1 and 3.2.

KnownDataFork: Because there isn't a TRsrcRec associated with the data fork, this routine uses special globals, gKnown and gInfected. The commentary on Globals' ProcessFile, below, discusses what kinds of files have their data forks fingerprinted.

LookForVirus nVIR: Did you know this virus exists in 2 versions? The May 1988 issue of MacTutor described a 372 byte long version, but when I asked Howard Upchurch to send me a copy of nVIR, the one he sent was 422 bytes long. That was the strain that got onto the CD-ROM disk. Later I picked up the 372 byte strain quite by accident at Kinko's. (I notified the Washington DC area Kinko's about this, incidentally.)

LookForVirus Scores: The first 12 bytes of the Scores virus CODE resource is used to hold the segment and jump table information, which vary according to which application it's infecting; therefore, we skip those 12 bytes in the fingerprinting process.

"Generic" Process xxx's: To shorten this article, only Process_ADBS is shown as an example of generic Process_xxx routines, which include ADBS CACH, CDEF, CODE, DRVr, DSAT, FKEY, FMTR, LDEF, MBDF, MDEF, MMAP, NBPC, PACK, PDEF, PTCH, ROv#, ROvr, SERD, WDEF, XCMD, XFCN, boot, cdev, mppc, snth and view (if it contains methods). The versions on the source code disk(s) include all of them except CODE, which would've been more work than I could do before the MacTutor publication deadline. Generic routines all call ProcessCurrRsrc to fingerprint the resource over its entire length and then test only for trusted ("Good") resources of that type.

Special Case Process xxx's: Routines that don't fingerprint over the entire length of the resource, or that also test for known viral ("Evil") resources, are published in this article.

Process DATA: Most innocuous DATA resources contain volatile data, so we default to the known flag to TRUE. The Scores DATA resource is simply a copy of the Scores CODE resource, so we also have to skip its first 12 bytes (see LookForVirus_Scores, above).

Process INIT: My suspicion that the Scores INITs might also store data internally was confirmed when different copies of the same INITs yielded different fingerprints. By analyzing raw memory dumps, I was able to determine that they skipped over variable parts by means of the hardware instruction BRA. We must also skip over these parts, or else our fingerprints will vary and we won't be able to recognize the INITs.

The test for \$60 is to detect the beginning of a BRA instruction. If the next byte is non-zero, it was a BRA.S, and we

have to skip over the number of bytes in that non-zero byte. If the next byte is zero, we have to skip over the number of bytes in the extension word that follows the zero byte.

To complicate the task of defense, future viruses may use other instructions to skip over variable data. Variable parts may be imbedded at different places in the resource, perhaps even in a variable way. If so, we'll just have to write tests for those situations too. Even if the virus succeeds at making the fingerprint method too complicated, we will always find, thru detailed analysis, other identifying characteristics that we can test for. The advantage of bending the fingerprint technique in this case is the relative certainty of identification, which counter-balances the dangers inherent in deleting resources based on appearance. Yes, I agree, all of this is very tedious, but the alternative is vulnerability.

Commentary on "Globals.p"

CONSTs: To look up the low memory global values in Inside Macintosh, omit the initial "k". The kIOBufferSize constant is used primarily by the KnownDataFork routine in Fingerprint.ipas. Under TML II, it's possible to use conditional compilation directives instead of the kProcessSelf constant, but the result wouldn't be TML 2.5 compatible (violating one of my goals). The kRsrc constants are arbitrary values used by InitRsrc, etc, below, to detect failure to call other Rsrc routines. This allows the program to display a complaint exactly diagnosing the problem, rather than going haywire by processing garbage. The values' only significance is that they're unlikely to occur randomly in memory. Just about any value other than 0 or -1 will do.

TYPEs: TCounts is used to keep counts in a consistent way; it's used mainly by the ListCounts routine, below. TFeedback is used to keep track of what was written to the screen and report file. JTE means "Jump Table Entry". TLoaded, TResIdOrIndex, TMainItem and TMainOpt are enumerations to make the code more readable; the TMain's give Main Dialog items descriptive names that can also be used in indexing. TRsrcRec is used to keep track of everything about a resource; I didn't want to undertake removing infected resources unless I kept a lot of information about them and handled them consistently (see InitRsrc, below). The TScores stuff is to get the JTE in bytes 4-11 (starting from 0) of the Scores CODE resource. TWordPtr is used just about everywhere.

VARs: gAbortPatrol and the gEvt stuff are used in small event loops in AbortPatrolIfCmdPeriodPressed and AwaitKeypress. gCode0 is used during PreprocessSelf in the main program and whenever the current file of the patrol contains CODE resources. gCounts and gTotals are used to keep counts within a patrol and overall, respectively. The gCurr variables track the current file, resource, etc of the patrol; most of them are passed to Globals by Patrol so it can know where we are in the patrol, but some are set within Globals. gDisabled and gOption are used to communicate with the user thru the Main Dialog; if gDisabled is set, the corresponding option cannot be changed by the user. gDlogPtr is used to keep track of the DialogPtr while the Main Dialog is hidden (usually during a patrol). gGrafPtr is used to save and restore TML Pascal's "plain vanilla"/"Textbook"

text window while the MainDialog is on the screen; to save processing, it's saved only once, in InitGlobals. gInd is used for consistent indentation. gSecsBegins and gSecsEnds are used in PatrolBegins and PatrolEnds to time the wall-clock duration of a patrol. gSfGetPt and gSfPutPt are used to center the SFGetFile and SFPutFile in the middle of the screen.

CallProcPtr: ProcessFile calls ProcessRsrcs with the address of the routine (a ProcPtr) to process each resource type; ProcessRsrcs has to use this INLINE to call that routine.

AbortPatrolIfCmdPeriodPressed: The major difference between this routine and the one that follows is the fact that AwaitKeypress stops the patrol until a key is pressed, while AbortPatrolIfCmdPeriodPressed checks for command-period on the fly. Both set the gAbortPatrol flag if command-period was pressed.

AwaitKeypress: This pauses the program (like REPEAT UNTIL KEYPRESS, which exists under TML 2.5 but not under TML II). In addition, it detects user request for cancellation with command-period.

Comment routines and Error routines: Error routines indent 2 deep (using gInd), comments 3. Error routines also have significance as far as beeping and awaiting are concerned.

DirectoryBegins: If there's something you want to do only once for a volume, such as check its boot blocks, this is a good place to do it, subordinate to an "IF (gCurrDirId = 2) THEN". (If gCurrDirId is 2, you're at the root directory of the volume.)

FixedCode0: The way a virus jumps back into applications gives us clues as to how we might repair them. So far, we've been lucky to have JTEs lying around to repair them with (using this routine). In the future, who knows? Unfortunately, there's no shortcut that doesn't involve disassembly to find out how to restore an application. If a new virus comes along, and if you don't do disassembly, but you want to code your own restoration code immediately, be sure to find out how to restore an application correctly from someone who has disassembled it. *[Fake jump table entries is something we all should be concerned about. -Ed]*

InitGlobals: This routine initializes the VARs defined above. Note that zeroing out a BOOLEAN sets it to FALSE and zeroing out a string sets it to the null string. The old way of centering a window or dialog was to use "screenBits.bounds", but that was from the days when everyone had only 1 screen. Since gGrafPtr contains the "plain vanilla"/"Textbook" window's GrafPtr, using gGrafPtr^.portBits.bounds will get the screen size of whatever screen it's in.

InitRsrc, GetRsrc, ReleaseRsrc and RemovedRsrc: These routines are a higher level interface to the Resource Manager. Their use cycle is InitRsrc, then GetRsrc, then, when you're done with it, ReleaseRsrc or RemovedRsrc. You may then return in a tighter circle to GetRsrc again. If you try to use these routines in any order other than this, they will report an error and abort the program. Also, if you want to see the program crash unpredictably, try releasing resources when processing Active Self and Active System.

JTEIsValid: TML 2.5 generates a longword compare with sign extension if you use \$A9F0, which doesn't compare cor-

rectly with the constant because it's not sign extended, hence the -22032.

LookForKnownViruses: This routine calls itself recursively during virus CODE removal because of the possibility of multiple infections. For example, an application could have CODEs 0 (jump table), 1 (application), 3 (Scores), 256 (nVIR) and 258 (Scores). As long as some infections are still being found and removed, I see no reason why the process should stop.

ProcessCodes: Sometimes an application contains a CODE 256, but isn't infected by nVIR. An example is PackIt. Note that PackIt also skips a CODE ResId, as an application infected with Scores does. Which CODE do you check to see if it's infected with Scores? Also, I'm told that MacMoney has a CODE resource that's exactly 7026 bytes long, like Scores, and that LaserSpeed installs atpl and DATA resources into your System file, like Scores.

These are the sorts of situations that made me start using fingerprinting in the first place, but still you have the problem of determining whether or not the application is actually infected. If a viral CODE resource is present but not linked-to by the jump table, then CODE 0 is not damaged, and you don't want to try to repair it! That would damage it! ProcessCodes calls the LookFor routines using the CODE resource pointed to by the first JTE. This greatly simplifies these otherwise confusing situations.

ProcessCodes also looks for irregularities in the jump table that might signal tampering. It warns the user if the jump table doesn't start with the first CODE resource, or if it ever descends (that could signal that an earlier resource was a new one added by a virus), or if it ever skips a resource ID as it ascends. (If the CODE resource pointed-to by the JTE stays the same, it avoids the test to see if JTEIsValid, which saves a considerable amount of I/O.) All applications infected by nVIR and/or Scores violate all 3 of these conditions.

Unfortunately, applications generated by Lightspeed often have jump tables that violate 2 or more of these conditions. I don't have Lightspeed, so I couldn't come up with an elegant way to deal with that. I'm open to suggestions. The program's inelegant solution is to have an exception list of applications whose jump tables are allowed to jump around all over the place. (The exception list uses the beginning of the application name in case there's a version number appended.) It's inelegant because they lose the benefit of the jump table irregularities search. They could get infected by some future virus and this program wouldn't detect anything suspicious in their jump tables.

ProcessFile: At present, the program fingerprints the data forks of only MacsBug and compiler object files. The latter is to guard against the possibility of a virus that doesn't go away even after recompilation, because its original infection source is an altered object file. RELBs are MDS-compatible ".Rel" files and OBJs are MPW-compatible ".o" files.

Also, at one time ROM patches were in the data fork of the System file. Now that there are PTCH resources, this may have been eliminated. If anyone knows whether or not the System file data fork can still contain executable code, please let everyone know by telling MacTutor. The same goes for executable code in any other, lesser-known file type's data fork, of course.

[Apple's system file data fork does indeed include executable code so this is another "danger" point in the system design that should be fingerprinted. -Ed]

A significant amount of code is to assure that we don't open and close the Active Self and Active System files. If you try treating them like any other file, the program will probably crash.

The NotePad File's data is in its data fork, and the Scrapbook File's data is in its resource fork. The Scores virus adds INIT resources and alters their Type and Creator to conscript them for use with the INIT 31 mechanism documented in IM IV-256. But it doesn't throw away their contents. (While infected, you're still able to use both DAs, because they continue to access those files by name.) That's why the program doesn't delete a file if either fork contains anything at all. The same deletion code will throw away the bogus "Desktop" and Scores files if it succeeds at removing all their infectious resources, incidentally.

Finally, you may ask, why don't we search for and destroy nVIR's "nVIR" resources? Well, earlier (during LookForKnownViruses) it may have detected an nVIR CODE resource but, for some reason, was unable to restore the application. So far, this hasn't ever happened, but if it ever did, the program would not have gotten to the code that removes the nVIR resources. One of those, nVIR 2, contains the JTE necessary to fix CODE 0. Because "nVIR" resources are not in themselves infectious, and because empty ones can be used to inoculate against nVIR (see the May 1988 MacTutor), you don't want to delete nVIR resources willy-nilly. The only time it's really safe to do so is after successfully restoring an infected application's CODE 0. This is true in only one place (at the end of Disinfected_nVIR).

ProcessRsrcs: Routines in Fingerprint.ipas decide which resources are infected. ProcessRsrcs takes that determination on faith and removes the resource if gOption[eRmVir] is on. (This is used as a wholesale resource type remover at the end of Disinfected_nVIR.)

Also, notice that we don't bump the index if a resource is removed. That's because the resource following it now assumes the removed resource's position in numerical order as far as Get1IndResource is concerned. This same consideration applies when a file is deleted. (See Patrol's CallProcessFile discussion of gCurrFileDeleted, below.)

ShortHexDump: This doesn't look right, but it is. Even though '0' is \$30 and 'A' is \$40, you have to add \$37 to get from 10 to 'A'.

Commentary on "MainDlog.p"

In order for SecurityPatrol to have no resource types except CODE, we have to build the Main Dialog's DITL in memory. It's not hard to do, keeps our private globals in synch with the dialog and is actually a lot of fun. Really.

VARs: The gHdl array and gDBtnRect are used to keep from having to call GetDItem all the time in MainDlogWorkRequested and FrameDefaultBtn, respectively. gDBtnRect is a Rect inset by 4 bits to the outside of the default button's Rect.

ChkChk: This sets the check boxes to the values represented by the gOption BOOLEANs. Note that the ORD of FALSE is 0 if and the ORD of TRUE is 1, a fact that's also used in building

the DITL in InitMainDlog (to keep the Dialog Items word-aligned).

FrameDefaultBtn: This routine frames the default button in accordance with the User Interface Guidelines chapter of Inside Macintosh.

InitMainDlog: This is the routine builds the DITL, calls NewDialog and sets up the private globals to speed up MainDlogWorkRequested. The sRect and sTitle local variable arrays are used to build the DITL with a FOR loop; this seems wordy, but actually uses fewer lines than doing them linearly, and is much more readable and maintainable.

KybdEquivsFilter: This routine is a filterProc to give every Main Dialog button and check box a keyboard equivalent.

MainDlogWorkRequested: This is the routine that actually manages the Main Dialog user interface. On entry, the dialog will always be hidden (InitMainDlog calls NewDialog with the "visible" parameter FALSE, and previous calls to will have hidden it before exiting. BringToFront is called before ShowWindow to keep the BringToFront from occurring visibly on screen and annoying the user; it doesn't hurt anything to have an invisible window momentarily the front window. The WHILE loop manages check boxes to free up the main program from having to worry about those details. The DITL was built with check boxes after buttons, so the WHILE loop will terminate when a Scope Of Work button is pushed. It then returns a TMainItem enumeration value (defined in Globals) to tell the main program which scope was requested, so that it'll know which Patrol routine to call.

Commentary on "Patrol.p"

Patrol is based on code examples that have already been published elsewhere, notably Tech Notes 24, 66, 68, 69 and 77, Inside Macintosh and earlier issues of MacTutor. It adds 2 new features: floating a Working Directory with the scan and keeping files in a directory together. Floating a Working Directory is to avoid overflowing 255 characters of partial pathname if the directories are deeply nested. (A virus creator might deliberately hide a virus deep in a series of nested folders to rely on other anti-virus utilities' possible reliance on full pathnames. Such a virus could very easily be launched by a document at the root level.) I've tested the floating Working Directory feature to 52 levels deep of nested folders, each with 31-character-long folder names and different filenames in each folder. As for keeping files in a directory together, previous directory scan algorithms were set up to recursively scan subdirectories right in the middle of the files being scanned, so you would see files from the parent directory, then files from a child directory, then more files from the parent directory, etc. Patrol doesn't go to subdirectories until it has patrolled all files in the current directory. This also minimizes opening and closing Working Directories.

kPatsInitd: This is an arbitrary value used by Patrol routines to detect failure to call InitPatrols. This allows the program to display a complaint exactly diagnosing the problem, rather than going haywire by processing garbage. The value's only significance is that it's unlikely to occur randomly in memory. Just about any value other than 0 or -1 will do.

TOverlappingPBs: For reasons unknown, Apple chose to

define a wide variety of parameter blocks with mostly-the-same/sometimes-different field names. For the limited uses in Patrol, the only major difference between HParamBlockRec in CIn-foPBlockRec is whether or not you're calling PBGetCatInfo. Rather than continually moving data around to keep their fields in synch, this TYPE allows defining PBs with which both types' field names can be used. This has been okay (so far) because the only PB values that Patrol uses happen to match up between the 2 types. Occasionally it can be a bit confusing, but it keeps field name usage usually correct.

VARs (private): gAppDirId and gAppVRefNum are used to detect "Active Self". gInitdFlag is used with kPatsInitd (see above) to detect failure to call InitPatrols. gOnly1Deep is used in PatrolDir to avoid processing subdirectories when the Scope Of Work is Directory (singular). gOrigWDRefNum is used to remember the wdRefNum of the first directory of the patrol (see FloatWDDeeper and FloatWDS shallower, below). gPBs is used to do the Low Level File Manager calls to do the patrol itself. gSFLst is passed to the SFGetFile dialogs in PatrolDirectories and PatrolFiles. gSysDirId and gSysVRefNum are used to detect "Active System". gWDPBRec is used by FloatWDDeeper and FloatWDS shallower to open and close working directories.

BuildDirname: This routine is called only when the current directory being patrolled changes. It uses a local variable of type TOverlappingPBs (see above) so as not to disturb the values in gPBs, which is still being used by the patrol. Starting from the new current directory (as pointed to by gCurrWDRefNum), it works backwards to the root directory of the volume; for example, if the new current directory is "A:B:C:", it would build "C:", then "B:C:", then "A:B:C:". ioFDirIndex is set to -1 because we're getting info about directories. (How to set ioFDirIndex is documented in Tech Note #69.) Initializing the loop by setting ioDrParID to 0 has no direct effect on the PBGetCatInfo call, because that's a field that's returned by the File Manager. Instead, it's being used in combination with "ioDrDirID := ioDrParID;" to set ioDrDirID to 0, which does affect the PBGetCatInfo call. (It will start the loop at the directory pointed-to by ioVRefNum, not some other directory on the same volume.) This may seem like a roundabout way to start getting folder names at gCurrWDRefNum, but the same line ("ioDrDirID := ioDrParID;") continues the loop upward thru the folder hierarchy, so it's actually pretty straightforward. The loop proceeds until ioDrDirID is 2, signifying the root directory of the volume. (Note that this works correctly even if gCurrWDRefNum points to a root directory, since the File Manager also returns ioDrDirID; the loop will have exactly one pass, beginning with ioDrDirID equal to 0 and ending with it equal to 2.) The routine simply terminates if an error occurs, such as gCurrDirname overflow. This is relatively acceptable because the program never uses the pathname to access a file during a patrol; instead, it uses gCurrWDRefNum and gCurrFilename to access the file, and treats gCurrDirname as commentary for the user. InitSecurityPatrol in the main program uses gCurrDirname in the ReWrite that creates the report file. This is unfortunately necessary because there isn't a parameter for a wdRefNum in the ReWrite statement. On the bright side, the user is unlikely to

place the report file more than 255 characters of full pathname deep.

CallProcessFile: Patrol centralizes all calls to ProcessFile via this routine to make absolutely sure that gActiveSelf and gActiveSys get set properly and that gCurrFileDeleted gets reset to FALSE. Correctly setting gActiveSelf and gActiveSys every time is vital to keep Security's ProcessFile routine from closing their resource forks. Resetting gCurrFileDeleted to FALSE every time is vital to prevent skipping the file following the one deleted, which would assume the numerical position of the deleted file within the patrol (pointed to by ioFDirIndex).

FloatWDDeeper and **FloatWDSshallower** (similarities): The problem with using pathnames to access files is the fact that you have only 255 characters to name it with. Since filenames and folder names can be up to 31 characters long, it's possible to overflow that limit with folders nested only 8 deep.. Patrol gets past this problem by "floating" a working directory. As a general rule, both of them close the working directory pointed-to by gCurrWDRefNum, use a DirId to open another working directory, and put the new wdRefNum into gCurrWDRefNum. The difference is what the newly created working directory represents.

FloatWDDeeper: This routine creates a working directory to move from a parent directory to a subdirectory using the DirId encountered in the patrol (namely, gPBs.fCPBRec.ioDrDirId).

FloatWDSshallower: This routine creates a working directory to return from a subdirectory to its parent. To do this, FloatWDSshallower could use an ioDrParId returned by PBGetCatInfo (see BuildDirname, above), but there's a much simpler method. PatrolDir calls itself recursively, so when it returns from patrolling a subdirectory, its pDrDirId parameter automatically reverts to the DirId of the parent. FloatWDSshallower uses this more readily available value.

FloatWDDeeper and **FloatWDSshallower** (exception): The only exception to the rule relates to the start directory of the patrol. If it's the root directory, PatrolDir starts out with a true yRefNum, not a wdRefNum. This is documented in Tech Note #77, which seemed to be warning us never to pass a vRefNum to PBCloseWD. (It didn't say so explicitly, but I haven't tried it to see what would happen.) To get around this problem, FloatWDDeeper makes an exception at the start of the floats (leaving the start directory open) and FloatWDSshallower makes an exception at the end (restoring the start directory with gOrigWDRefNum).

GetActualDirId: This routine assures that gCurrDirId will always be accurate, even when PatrolDir is called with pDrDirId equal to 0. Even though Globals doesn't currently use gCurrDirId, Patrol uses it to detect the Active Self and Active System.

InitPatrols: This routine makes sure that all of Patrol's globals are properly initialized. The ZeroFillRange calls implicitly set all BOOLEAN variables to FALSE, all strings to the null string and all ioCompletion fields to NIL. Using kFSFCBLen to see if HFS is active is documented in Tech Note #66. Calling GetVRefNum to get gSysVRefNum is documented in Tech Note #77. Calling PBHGetVInfo and referencing ioVFndrInfo[1] to

get gSysDirId is documented in Tech Note #67. Referencing kBootDrive instead if HFS is not active is also in #77.

PatrolDir: This routine is the heart and soul of Patrol. Its techniques are documented in Tech Note #69. If you understand that, you'll understand this, but PatrolDir's way of doing things is considerably different. It patrols files first (ignoring subdirectories), then patrols subdirectories (ignoring files). Patrolling files uses PBGetFInfo (note: not PBHGetFInfo). Patrolling subdirectories uses PBGetCatInfo. There are three major advantages of doing it that way:

- (1) filenames appear in an unbroken list,
- (2) the program can work even if HFS isn't active,
- (3) it's easier to patrol only 1 directory deep.

There are other differences from Tech Note #69: It calls FloatWDDeeper, FloatWDSshallower and GetActualDirId, for reasons discussed above. Also, if gError <> NoErr after a recursive call, it backs out of the recursion rapidly until it reaches the start directory of the patrol; this saves a lot of unnecessary calls to FloatWDSshallower, since doing only the last one has the same effect as doing them all.

PatrolFiles: I've gone to considerable trouble here to correctly call DirectoryEnds and DirectoryBegins whenever the user selects a file in a different directory from the previously selected file. That's the purpose of the imbedded procedure CallPrevDirEnd. If you don't go to that trouble, the TFeedback variables in Globals won't get reset properly and the new directory name won't get displayed on the screen nor in the report file.

Commentary on "SecurityPatrol.p"

SecurityPatrol uses standard Pascal text I/O to report its findings and progress to the user. (In TML terminology, the TML 2.5 version is called a "plain vanilla application" and the TML II version is called a "Textbook application".) The main program also manages initialization, termination and high-level interface between MainDlog and Patrol.

Because there's no event loop, the program doesn't support DAs (on-purpose). It also specifically disables FKEYs. The only form of concurrent code that it doesn't disallow is MultiFinder, but you shouldn't run it under MultiFinder anyway, because you won't be able to examine open files, such as the DeskTop file and the concurrent applications themselves (the Finder, eg).

There's a very good reason why I don't like the idea of concurrently executing code. Let me give you a hypothetical scenario: Suppose someone imbedded a virus in a WDEF and installed that WDEF into the DeskTop file of a disk. I don't know how the Finder does things, but suppose it allows a runtime override of definition procedures using the standard Resource Manager precedence (document → application → System file). You're patrolling files in foreground under MultiFinder, you get an SFGGetFile dialog to patrol Directories, and you insert the infected disk. In background, the Finder reads the DeskTop file of the disk *and displays the disk's window behind the SecurityPatrol window using the override WDEF*. You've just been infected, and SecurityPatrol hasn't even had a chance to look at the disk yet. My guess is that the published version of SecurityPatrol wouldn't find anything on that disk. Whether or not your version would is up to you.

Of course, that scenario relies on a presupposition that the Finder allows a runtime override of WDEFs, which it probably doesn't do. On the other hand, I don't know *for sure* that it doesn't, and I don't like the uncontrolled variable it represents. Even if the Finder wouldn't let a virus thru this way, maybe some other concurrent application or DA would. I didn't go extraordinarily out of my way to prevent all concurrent code (VBLs, AppleTalk, etc), but I suspect there's a security hole there somewhere. I'm voicing my concern here to focus more minds than my own on the subject. *[Apple has also thought of this problem and is working on it presently. They just don't want the wrong kind of minds to be focused on it, if you know what I mean! -Ed]*

“(OUTPUT)”: The presence of this phrase in the program header tells TML 2.5 that this is a “plain vanilla” application. Before the main procedure starts to execute, the runtime library will initialize all the managers it needs for text I/O, create the WriteLn window and put “TML Pascal” into its drag bar title. Because it initializes the managers, we don't. It's ignored by TML II.

kPreprocessSelf and **kAwaitVerification**: Having kPreprocessSelf turned on can get to be a real nuisance during development, but it's an important safeguard and better than processing Active Self during the patrols. You may want to turn it off during development and back on again in the final cleanup before release to your friends and/or company's users. The code controlled by kAwaitVerification will be cleaned up in Version 2.0: All of SecurityPatrol's own CODE resources except the one containing Fingerprint.ipas should be fingerprinted and tested in Fingerprint.ipas. The verification task presented to the user could then be made much simpler.

ScrDmpEnbPtr and **ScrDmpEnbSave**: These are used in InitSecurityPatrol and ExitSecurityPatrol to save, disable and restore FKEYs.

[\$Z*]: A TML 2.5 to TML II upgrade issue not mentioned in the TML II manual's Appendix F is the fact that main program's routine names, when referenced from a UNIT, will not be found at Link time unless you explicitly define them externally with this directive. It's documented in Appendix C, but not as a difference.

InitSecurityPatrol: Textbook is a TML II routine in PasLibIntf, distributed by TML Systems with the compiler. It initializes all the managers it needs for text I/O, creates the WriteLn window and puts the application's name into its drag bar title. Without it TML II would WriteLn all over the desktop. Because it initializes the managers, we don't. For TML 2.5 users, a dummy version of Textbook is defined in the PasLibIntf.Pas UNIT on the source code disk(s).

The initial values of the Main Dialog check boxes are set between the calls to InitGlobals and InitMainDlog.

When AppleTalk is not active, the statement “ReWrite(o,'PRINTER:’)” is used to send the report file to the printer. Under TML 2.5, this prints to a direct-connect ImageWriter on serial port B in streaming text mode, which is even more draft than draft mode (the ImageWriter doesn't recognize curly quotes, for example). Under TML II, this

feature doesn't work, and the report file output will be, in effect, thrown away. Tom Leonard is working on the problem. In the meantime, you can save the report to a text file and print it later with MPW, which has the distinct advantage of also being able to print to a LaserWriter.

TML 2.5 allows TextFace calls to affect the WriteLn output. TML II doesn't, but it doesn't hurt.

PreprocessSelf: These are the most stringent tests I could think of to guard against the program's own infection. You're invited to add more. If you leave in the line that turns on gOption[eRmVir], the program will disinfect itself of all viruses it knows how to remove. If you take out that line, it will detect known viruses and abort, so you should tell your users always to run it from a write protected disk.

TML 2.5 Link puts a JMP d(PC) instruction at the beginning of CODE 1 that jumps into the actual start address of SecurityPatrol. That's why it checks for \$4EFA and offsets by the extension word that follows. TML II is unaffected by this test.

Wryte routines: These routines manage text I/O. Under TML 2.5, Write, WriteLn, etc, can only appear in the main program, not in UNITs. This limitation was done away with in TML II, but this way of doing things is compatible. Moreover, by centralizing output, it allows calling PLFlush on every WriteLn.

PLFlush is a TML II routine in PasLibIntf. (See “(OUTPUT)”, above.) It flushes a file's buffer. “PLFlush(OUTPUT);” assures that screen feedback will be current. Alternatively, you could call “PLSetVBuf(OUTPUT,NIL,\$40,256);” in InitSecurityPatrol, but I like this stick-shift level of control, and it seems to work a little better.

TML 2.5 used to print numbers with a default field width of 1. TML II now uses the ANSI-standard Pascal default field width (6, I think). To avoid inconsistent results between the 2 versions of the program, WryteNbr never uses the default field width, but instead specifies field width explicitly.

zzSecurityPatrol: See the end of the commentary on CodeSizeLimits.p, above.

The source code disk for this issue (see the MacTutor Mail Order Store Ad) contains both the TNL 2.5 and TML II versions with the complete system file resource fingerprints for previous system files. We recommend purchase of this disk, only \$8!

(Copyright Header

©1988 by Steve Seaquist. All rights reserved. Used by permission. Use at your own risk. No warranty is expressed or implied. Neither Apple Computer nor MacTutor endorse or warrant this program in any way, nor are they responsible for its use or mis-use in any way.

This Macintosh virus-detecting program was originally published and explained in the February 1989 issue of MacTutor magazine. Some aspects of its design are important to security, and it uses some unusual techniques, so please read the article.)

“CodeSizeLimits.p”

UNIT CodeSizeLimits;

```

INTERFACE
VAR
  gJtSize:      INTEGER;
  gEntryPoint:  LONGINT;
  gSizeLimit:   ARRAY [0..8] OF LONGINT;
  gMaxCode:     INTEGER;
PROCEDURE GetCodeSizeLimits;
IMPLEMENTATION
PROCEDURE zzSecurityPatrol;      EXTERNAL;
PROCEDURE GetCodeSizeLimits;
BEGIN
  gEntryPoint := ORD4(@zzSecurityPatrol)+$1A;
  gJtSize     := 1240;
  gMaxCode    := 8;
  gSizeLimit[0] := gJtSize + 16;
  gSizeLimit[1] := 15700;
  gSizeLimit[2] := 23900;
  gSizeLimit[3] := 11200;
  gSizeLimit[4] := 00844;
  gSizeLimit[5] := 01908;
  gSizeLimit[6] := 01606;
  gSizeLimit[7] := 01822;
  gSizeLimit[8] := 01312;
END;
END.

"Fingerprint.ipas"
(File Fingerprint.ipas)
VAR
  gFgPr1,gFgPr2,gFgPr3,gFgPr4,gFgPr5,gFgPr6,gFgPr7: LONGINT;
PROCEDURE CommentFgPr;
BEGIN
  Writeln ('(');
  Writeln (gFgPr1,5);
  Writeln (',');
  Writeln (gFgPr2,9);
  Writeln (',');
  Writeln (gFgPr3,6);
  Writeln (',');
  Writeln (gFgPr4,5);
  Writeln (',');
  Writeln (gFgPr5,5);
  Writeln (',');
  Writeln (gFgPr6,5);
  Writeln (',');
  Writeln (gFgPr7,9);
  Writeln (')');
END;
PROCEDURE CommentFgPrData;
BEGIN
  ErrorBegins('Unknown data fork');
  CommentFgPr;
  ErrorEnds(0);
END;
PROCEDURE CommentFgPrRsrc(pRsrcPtr: TRsrcPtr);
BEGIN
  IF (gFgPrTitle <> '') THEN
  BEGIN
    ErrorMessage(gFgPrTitle,0);
    gFgPrTitle := '';
  END;
  CommentRsrcBegins(pRsrcPtr);
  CommentFgPr;
  Writeln;
END;
FUNCTION Evil (Edited so that ... )
(p1,p2,p3: LONGINT) ( ... calls fit into ... )
:      BOOLEAN; ( ... MacTutor column... )
BEGIN ( ... width. Source ... )
  Evil := FALSE; ( ... code disk ... )
  IF (gFgPr1 = p1) ( ... version tests... )
  AND (gFgPr2 = p2) ( ... all 7 gFgPr's. )
  AND (gFgPr3 = p3) THEN
  WITH gCurrRsrc DO

```

```

BEGIN
  Evil      := TRUE;
  fKnown    := TRUE;
  fInfected := TRUE;
  gInfected := TRUE;
END;
END;
PROCEDURE FgPr(pPtr: Ptr; pSize: Size);
VAR
  i,sTemp:      LONGINT;
  sWordPtr:     TWordPtr;
PROCEDURE FgPrTemp;
BEGIN
  gFgPr2 := gFgPr2 + sTemp;
  gFgPr3 := Bxor(gFgPr3,sTemp);
  gFgPr4 := gFgPr4 + ORD(ODD(sTemp));
  gFgPr5 := gFgPr5 + ORD(sTemp< 0);
  gFgPr6 := gFgPr6 + ORD(sTemp> 0);
  IF ODD(i) THEN
    gFgPr7 := gFgPr7 + sTemp
  ELSE
    gFgPr7 := gFgPr7 - sTemp;
  END;
END;
BEGIN
  sWordPtr := TWordPtr(pPtr);
  FOR i := 1 TO (pSize DIV 2) DO
  BEGIN
    sTemp := ORD4(sWordPtr^);
    FgPrTemp;
    INC(LONGINT(sWordPtr));
    INC(LONGINT(sWordPtr));
  END;
  IF ODD(pSize) THEN
  BEGIN
    sTemp := BAnd(ORD4(sWordPtr^),$FF00);
    FgPrTemp;
  END;
  gFgPr1 := gFgPr1 + pSize;
END;
FUNCTION Good (Edited so that ... )
(p1,p2,p3: LONGINT) ( ... calls fit into ... )
:      BOOLEAN; ( ... MacTutor column... )
BEGIN ( ... width. Source ... )
  Good := FALSE; ( ... code disk ... )
  IF (gFgPr1 = p1) ( ... version tests... )
  AND (gFgPr2 = p2) ( ... all 7 gFgPr's. )
  AND (gFgPr3 = p3)
  AND (gFgPr4 = p4)
  AND (gFgPr5 = p5)
  AND (gFgPr6 = p6)
  AND (gFgPr7 = p7) THEN
  WITH gCurrRsrc DO
  BEGIN
    Good      := TRUE;
    fKnown    := TRUE;
    fInfected := FALSE;
  END;
END;
PROCEDURE InitFgPr;
BEGIN
  gFgPr1 := 0;
  gFgPr2 := 0;
  gFgPr3 := 0;
  gFgPr4 := 0;
  gFgPr5 := 0;
  gFgPr6 := 0;
  gFgPr7 := 0;
END;
FUNCTION KnownDataFork: BOOLEAN;
VAR
  sBytesRemaining: LONGINT;
  sBytesThisPass: LONGINT;
PROCEDURE KnownIF
(p1,p2,p3,p4,p5,p6,p7: LONGINT);

```

```

BEGIN
  IF (gFgPr1 = p1)
  AND (gFgPr2 = p2)
  AND (gFgPr3 = p3) THEN
    EXIT(KnownDataFork);
  END;
BEGIN
  KnownDataFork := TRUE;
  InitFgPr;
  sBytesRemaining := gCurrEOF;
  WHILE sBytesRemaining > 0 DO
    BEGIN
      IF (sBytesRemaining > kIOBufferSize) THEN
        sBytesThisPass := kIOBufferSize
      ELSE
        sBytesThisPass := sBytesRemaining;
      gError := FSRead(gCurrRefNum,
        sBytesThisPass,
        gCurrIOBuffer);
      IF (gError <> NoErr) THEN
        BEGIN
          ErrorOSErr('Couldn't read data fork');
          EXIT(KnownDataFork);
        END;
      FgPr(gCurrIOBuffer, sBytesThisPass);
      sBytesRemaining :=
        sBytesRemaining - sBytesThisPass;
    END;
    KnownIF( 512, 1888894, 23566); (DRVRRuntime)
    KnownIF(34304, 122743292, -30376); (Interface)
    KnownIF( 2560, 9884035, -13755); (ObjLib)
    KnownIF( 8704, 38905655, -5215); (PerformLib)
    KnownIF(22016, 115306063, 12985); (Runtime)
    KnownIF( 6656, 28028983, -22709); (ToolLibs)
    KnownIF( 3072, 12526030, -25732); (HyperXCMD)
    KnownIF( 2560, 11236115, -31819); (SANELib)
    KnownIF( 4608, 16761354, -16062); (SANELib881)
    KnownIF(15872, 62109326, -18394); (TMLPasLib)
    KnownIF(27634, 204503778, -8384); (MacBug 5.5)
    KnownDataFork := FALSE;
  END;
  PROCEDURE LookForVirus_nVIR;
  BEGIN
    IF (gCurrRsrc.fResId <> 256) THEN
      EXIT(LookForVirus_nVIR);
    gCurrRsrc.fInfected := TRUE;
    ProcessCurrRsrc;
    IF Evil( 372, 1334566, -15078) THEN EXIT;
    IF Evil( 422, 1445005, -22775) THEN EXIT;
  END;
  PROCEDURE LookForVirus_Scores;
  BEGIN
    IF (gCurrRsrc.fSize < 7026) THEN
      EXIT(LookForVirus_Scores);
    InitFgPr;
    FgPr(Ptr(ORD4(gCurrRsrc.fHdl^)+12), 7014);
    IF gOption[eFgPr] THEN
      BEGIN
        gFgPrTitle := 'Short fingerprint';
        CommentFgPrRsrc(@gCurrRsrc);
      END;
    IF Evil( 7014, 32071691, 16777) THEN EXIT;
  END;
  PROCEDURE ProcessCurrRsrc;
  BEGIN
    InitFgPr;
    FgPr(gCurrRsrc.fHdl^, gCurrRsrc.fSize);
  END;
  PROCEDURE ProcessRemoveRsrc;
  BEGIN
    gCurrRsrc.fKnown := TRUE;
    gCurrRsrc.fInfected := TRUE;
  END;
  PROCEDURE Process_ADBS;

```

```

BEGIN
  ProcessCurrRsrc;
  IF Good( 262, 946915, -549) THEN EXIT;
END;
PROCEDURE Process_DATA;
BEGIN
  WITH gCurrRsrc DO
    BEGIN
      fKnown := TRUE;
      IF (fSize < 7026) THEN
        EXIT(Process_DATA);
      InitFgPr;
      FgPr(Ptr(ORD4(fHdl^)+12), 7014);
      IF gOption[eFgPr] THEN
        BEGIN
          gFgPrTitle := 'Short fingerprint';
          CommentFgPrRsrc(@gCurrRsrc);
        END;
      END;
    IF Evil( 7014, 32071691, 16777) THEN EXIT;
  END;
  PROCEDURE Process_INIT;
  VAR
    sOffset: INTEGER;
    sPtr: Ptr;
  BEGIN
    WITH gCurrRsrc DO
      BEGIN
        InitFgPr;
        IF (fResId = 6)
        OR ((fResId = 10)
        AND (gCurrFilename <> 'Vaccine'))
        OR (fResId = 17) THEN
          BEGIN
            sOffset := 0;
            sPtr := fHdl^;
            IF (sPtr = $60) THEN
              BEGIN
                INC(LONGINT(sPtr));
                IF (sPtr = 0) THEN
                  BEGIN
                    INC(LONGINT(sPtr));
                    sOffset := 2 + TWordPtr(sPtr)^;
                  END
                ELSE
                  sOffset := 2 + sPtr^;
              END;
            FgPr(Ptr(ORD4(fHdl^)+sOffset),
              fSize-sOffset);
            IF gOption[eFgPr] THEN
              BEGIN
                gFgPrTitle := 'Short fingerprint';
                CommentFgPrRsrc(@gCurrRsrc);
              END;
            END
          ELSE
            FgPr(fHdl^, fSize);
          END;
        IF Good( 2234, 5918345, 24783) THEN EXIT;
        IF Good( 2, 20085, 20085) THEN EXIT;
        IF Good( 580, 2390534, -3964) THEN EXIT;
        IF Good( 256, 624295, -11467) THEN EXIT;
        IF Good( 276, 821588, -22226) THEN EXIT;
        IF Good( 318, 1106224, 15474) THEN EXIT;
        IF Good( 372, 1325194, 13502) THEN EXIT;
        IF Good( 262, 889886, -25106) THEN EXIT;
        IF Good( 5200, 16204911, -21859) THEN EXIT;
        IF Good( 514, 2113878, -13890) THEN EXIT;
        IF Good( 264, 920108, -860) THEN EXIT;
        IF Good( 436, 1746895, -3577) THEN EXIT;
        IF Good( 358, 1167886, 15360) THEN EXIT;
        IF Good( 26, 102477, -9939) THEN EXIT;
        IF Good( 944, 3348718, -13868) THEN EXIT;
        IF Good( 820, 2799073, -29445) THEN EXIT;

```

```

IF Good( 572, 1840164, -5246) THEN EXIT;

IF Evil( 366, 1333180, -29162) THEN EXIT;
IF Evil( 416, 1443619, -5115) THEN EXIT;
IF Evil( 758, 3291138, 6608) THEN EXIT;
IF Evil( 1014, 4600985, 19785) THEN EXIT;
IF Evil( 474, 1932041, 18387) THEN EXIT;
END;

```

```

PROCEDURE Process_atpl;

```

```

BEGIN

```

```

ProcessCurrRsrc;

```

```

IF Good( 4874, 17745131, 2851) THEN EXIT;

```

```

IF Evil( 2410, 10235053, -25635) THEN EXIT;
END;

```

"Globals.p"

```

UNIT Globals;

```

```

INTERFACE

```

```

USES

```

```

MemTypes, QuickDraw, OSIntf, ToolIntf, PackIntf;

```

```

CONST

```

```

(— Low Mem Globals —)

```

```

kCurApName = $910;

```

```

kCurApRefNum = $900;

```

```

kBootDrive = $210;

```

```

kResLoad = $A5E;

```

```

kScrDmpEnb = $2F8;

```

```

kSFCBLen = $3F6;

```

```

kSPConfig = $1FB;

```

```

kSysMap = $A58;

```

```

kSysResName = $AD8;

```

```

(— Other constants —)

```

```

kIOBufferSize = 10000;

```

```

kProcessSelf = FALSE;

```

```

kRsrcHdlValid = 9876543;

```

```

kRsrcIsInitd = 3456789;

```

```

kZeroOutVirs = TRUE;

```

```

TYPE

```

```

TCountsPtr = ^TCountsRec;

```

```

TCountsRec =

```

```

RECORD

```

```

fDeleted: LONGINT;

```

```

fExamined: LONGINT;

```

```

fFiles: LONGINT;

```

```

fInfected: LONGINT;

```

```

fRemoved: LONGINT;

```

```

fResources: LONGINT;

```

```

END;

```

```

TFeedbackPtr = ^TFeedbackRec;

```

```

TFeedbackRec =

```

```

PACKED RECORD

```

```

fWroteDirname: BOOLEAN;

```

```

fWroteFilename: BOOLEAN;

```

```

END;

```

```

TJTEHdl = ^TJTEPtr;

```

```

TJTEPtr = ^TJTERec;

```

```

TJTERec =

```

```

RECORD

```

```

fOffset: INTEGER;

```

```

fSkip3F3C: INTEGER;

```

```

fSegId: INTEGER;

```

```

fSkipA9F0: INTEGER;

```

```

END;

```

```

TJTHdl = ^TJTPtr;

```

```

TJTPtr = ^TJTRec;

```

```

TJTRec =

```

```

RECORD

```

```

fAboveA5Size: LONGINT;

```

```

fBelowA5Size: LONGINT;

```

```

fNbrBytesInTable: LONGINT;

```

```

fTableOffset: LONGINT;

```

```

fJTEEntry:

```

```

ARRAY [1..1] OF TJTERec;

```

```

END;

```

```

TLoaded = (eNotYet, eAlreadyLoaded, eWeLoadedIt);

```

```

TMainItem =

```

```

(eNotADLogItem,

```

```

eDirs, eDiry, eEvery, eFiles, eQuit,

```

```

eAwait, eBeeps, eFgPr, eFgPrC,

```

```

eLList, eRmVir, eTrace,

```

```

eMain, eOpts, eScOW,

```

```

eDBtn);

```

```

TMainOpt = ARRAY [eAwait..eTrace] OF BOOLEAN;

```

```

TPaocRec = PACKED ARRAY [1..1] OF CHAR;

```

```

TResIdOrIndex = (ResId, Index);

```

```

TRsrcPtr = ^TRsrcRec;

```

```

TRsrcRec =

```

```

RECORD

```

```

fFlag: LONGINT;

```

```

fHdl: Handle;

```

```

fInfected: BOOLEAN;

```

```

fKnown: BOOLEAN;

```

```

fLoaded: TLoaded;

```

```

fResAttrs: INTEGER;

```

```

fResId: INTEGER;

```

```

fResType: ResType;

```

```

fSize: Size;

```

```

fState: SignedByte;

```

```

END;

```

```

TScoresHdl = ^TScoresPtr;

```

```

TScoresPtr = ^TScoresRec;

```

```

TScoresRec =

```

```

RECORD

```

```

fOffsetToFirstJTE: INTEGER;

```

```

fNbrJTEsForRsrc: INTEGER;

```

```

fOldJTE: TJTERec;

```

```

END;

```

```

TWordHdl = ^TWordPtr;

```

```

TWordPtr = ^INTEGER;

```

```

VAR

```

```

gAAGlobals: SignedByte;

```

```

gAbortPatrol, gActiveSelf, gActiveSys: BOOLEAN;

```

```

gCode0: TRsrcRec;

```

```

gCounts: TCountsRec;

```

```

gCurrDInfo: DInfo;

```

```

gCurrDirId, gCurrEOF: LONGINT;

```

```

gCurrDirname: Str255;

```

```

gCurrIOBuffer: Ptr;

```

```

gCurrFileDeleted: BOOLEAN;

```

```

gCurrFilename: Str255;

```

```

gCurrFInfo: FInfo;

```

```

gCurrIndex: INTEGER;

```

```

gCurrRefNum: INTEGER;

```

```

gCurrRsrc: TRsrcRec;

```

```

gCurrVRefNum: INTEGER;

```

```

gCurrWRefNum: INTEGER;

```

```

gDateTimeRec: DateTimeRec;

```

```

gDisabled: TMainOpt;

```

```

gDlgPtr: DialogPtr;

```

```

gError: OSErr;

```

```

gEvt: EventRecord;

```

```

gEvtMask: INTEGER;

```

```

gFgPrTitle: Str255;

```

```

gGrafPtr: GrafPtr;

```

```

gHFS: BOOLEAN;

```

```

gInd: STRING[10];

```

```

gInfected: BOOLEAN;

```

```

gInfectedWritten: BOOLEAN;

```

```

gOption: TMainOpt;

```

```

gPgmname: Str255;

```

```

gReportFlags: TFeedbackRec;

```

```

gScreenFlags: TFeedbackRec;

```

```

gSecsBegins: LONGINT;

```

```

gSecsEnds: LONGINT;

```

```

gSFGetPt: Point;

```

```

gSFPutPt: Point;

```

```

gSFRep: SFReply;

```

```

gTotals: TCountsRec;

```



```

FUNCTION Disinfected_nVIR: BOOLEAN;
VAR
  snVIR2: TRsrcRec;
  sCodeGone: BOOLEAN;
BEGIN
  IF gOption[eTrace] THEN
    Trace('Disinfected_nVIR');
  Disinfected_nVIR := FALSE;
  InitRsrc(@snVIR2);
  GetRsrc (@snVIR2, 'nVIR', 2, ResId);
  WITH snVIR2 DO
    BEGIN
      IF (fFlag <> kRsrcHdlValid) THEN
        BEGIN
          ErrorInfected('No nVIR 2!');
          ReleaseRsrc(@gCurrRsrc);
          EXIT(Disinfected_nVIR);
        END;
      IF (fSize < 8) THEN
        BEGIN
          ErrorInfected('Too small nVIR 2!');
          ReleaseRsrc(@gCurrRsrc);
          ReleaseRsrc(@snVIR2);
          EXIT(Disinfected_nVIR);
        END;
      MoveHHi(fHdl);
      HLock (fHdl);
      IF NOT(FixedCode0(TJTEPtr(fHdl))) THEN
        BEGIN
          ReleaseRsrc(@gCurrRsrc);
          ReleaseRsrc(@snVIR2);
          EXIT(Disinfected_nVIR);
        END;
      Disinfected_nVIR := TRUE;
      sCodeGone := RemovedRsrc(@gCurrRsrc);
      ReleaseRsrc(@snVIR2);
      ProcessRsrcs('nVIR', @ProcessRemoveRsrc);
      IF sCodeGone
      AND (Count1Resources('nVIR') = 0) THEN
        ErrorMessage('nVIR removed', 0)
      ELSE
        BEGIN
          ErrorMessage('nVIR "disinfected":', 0);
          CommentBegins;
          Wryte ('All of its resources are now ');
          Wryte ('harmless, but some were not ');
          WryteLn('removed, for some reason. ');
        END;
      END;
    END;
  END;
END;
FUNCTION Disinfected_Scores: BOOLEAN;
BEGIN
  IF gOption[eTrace] THEN
    Trace('Disinfected_Scores');
  Disinfected_Scores := FALSE;
  WITH gCurrRsrc DO
    BEGIN
      MoveHHi(fHdl);
      HLock (fHdl);
      WITH TScoresHdl(fHdl)^ DO
        IF NOT(FixedCode0(@f01dJTE)) THEN
          BEGIN
            ReleaseRsrc(@gCurrRsrc);
            EXIT(Disinfected_Scores);
          END;
        Disinfected_Scores := TRUE;
        IF RemovedRsrc(@gCurrRsrc) THEN
          ErrorMessage('Scores removed', 0)
        ELSE
          ErrorMessage('Scores disinfected', 0);
        END;
      END;
    END;
  END;
  PROCEDURE ErrorBegins(pStr: Str255);
  BEGIN

```

```

    WryteFilename;
    Wryte (gInd);
    Wryte (gInd);
    Wryte (pStr);
  END;
  PROCEDURE ErrorEnds(pBeeps: INTEGER);
  VAR
    i, sBeeps: INTEGER;
  BEGIN
    IF gOption[eBeeps] THEN
      BEGIN
        IF (pBeeps > 4) THEN
          sBeeps := 4
        ELSE
          sBeeps := pBeeps;
        FOR i := 1 TO sBeeps DO
          SysBeep(3);
        END;
      END;
    IF gOption[eAwait] THEN
      BEGIN
        WryteLn(' (WAITING ON KEY PRESS)');
        AwaitKeyPress;
      END;
    ELSE
      WryteEoln;
    END;
  PROCEDURE ErrorInfected(pStr: Str255);
  BEGIN
    IF NOT(gInfectedWritten) THEN
      BEGIN
        ErrorBegins('**!INFECTED!** ');
        WryteEoln;
        gInfectedWritten := TRUE;
      END;
    IF (pStr <> '') THEN
      BEGIN
        CommentBegins;
        Wryte(pStr);
        ErrorEnds(3);
      END;
    END;
  END;
  PROCEDURE ErrorMessage(pStr: Str255; pBeeps: INTEGER);
  BEGIN
    ErrorBegins(pStr);
    ErrorEnds(pBeeps);
  END;
  PROCEDURE ErrorOSErr(pStr: Str255);
  BEGIN
    IF (pStr <> '') THEN
      BEGIN
        ErrorBegins(pStr);
        WryteEoln;
      END;
    CommentBegins;
    Wryte ('OSErr code = ');
    WryteNbr(gError, 1);
    ErrorEnds(2);
  END;
  FUNCTION FixedCode0(pJTPtr: TJTEPtr): BOOLEAN;
  BEGIN
    FixedCode0 := FALSE;
    IF gOption[eTrace] THEN
      Trace('FixedCode0');
    IF NOT(JTEIsValid(pJTPtr)) THEN
      BEGIN
        ErrorInfected('Bad Jump Table Entry!');
        EXIT(FixedCode0);
      END;
    IF NOT(gOption[eRmVir]) THEN
      BEGIN
        ErrorInfected('Remove option off');
        CommentBegins;
        WITH pJTPtr^ DO
          BEGIN

```

```

Writeln ('Jumps to ');
Writeln (fOffset,1);
Writeln (' of CODE ');
Writeln (fSegId,1);
Writeln;
END;
ErrorMsg('Not removed',1);
EXIT(FixedCode0);
END;
WITH gCode0 DO
BEGIN
IF gOption[eTrace] THEN
BEGIN
Trace('About to restore CODE 0');
AbortPatrolIfCmdPeriodPressed;
IF gAbortPatrol THEN
EXIT(FixedCode0);
END;
TJTHd1(fHd1)^^.fJTEntry[1] := pJTPtr^;
IF (BAnd(fResAttrs,resProtected) <> 0)
AND (fResAttrs <> -1) THEN
BEGIN
SetResAttrs(fHd1,0);
ChangedResource(fHd1);
gError := ResError;
SetResAttrs(fHd1,fResAttrs);
END
ELSE
BEGIN
ChangedResource(fHd1);
gError := ResError;
END;
IF (gError <> NoErr) THEN
BEGIN
ErrorInfected('CODE 0 unchanged!');
IF (gError = wPrErr) THEN
ErrorMsg('Disk is locked',0)
ELSE
ErrorOSerr('');
gError := 0;
EXIT(FixedCode0);
END;
WriteResource(fHd1);
gError := ResError;
IF (gError <> NoErr) THEN
BEGIN
ErrorInfected('CODE 0 unwritten!');
ErrorOSerr('');
EXIT(FixedCode0);
END;
END;
FixedCode0 := TRUE;
END;
PROCEDURE GetRsrc
(pRsrcPtr: TRsrcPtr;
pResType: ResType;
pInt: INTEGER;
pIntIs: TRsrcIDOrIndex);
VAR
sName: Str255;
sResLoad: BOOLEAN;
PROCEDURE CommentWhich;
BEGIN
CommentBegins;
WritelnType(pResType);
WritelnChar(' ');
Writeln (pInt,1);
IF (pIntIs = Index) THEN
Writeln (' (indexed)');
Writeln;
END;
BEGIN
WITH pRsrcPtr^ DO
BEGIN

```

```

IF (fFlag <> kRsrcIsInitd) THEN
BEGIN
ErrorMsg('Logic error using GetRsrc',4);
AwaitKeyPress;
ExitSecurityPatrol;
END;
fResType := pResType;
fResId := pInt;
sResLoad := (TWordPtr(kResLoad) <> 0);
IF (gActiveSelf OR gActiveSys) THEN
SetResLoad(FALSE);
IF (pIntIs = Index) THEN
BEGIN
IF gOption[eTrace] THEN
TraceRsrc('About to get ind',pRsrcPtr);
fHd1 := Get1IndResource(pResType,pInt);
END
ELSE
BEGIN
IF gOption[eTrace] THEN
TraceRsrc('About to get',pRsrcPtr);
fHd1 := Get1Resource(pResType,pInt);
END;
IF sResLoad THEN
BEGIN
IF (gActiveSelf OR gActiveSys) THEN
SetResLoad(TRUE);
IF (fHd1 = NIL)
OR (ORD4(fHd1) = -1) THEN
BEGIN
gError := ResError;
ErrorOSerr('Couldn't get resource');
CommentWhich;
InitRsrc(pRsrcPtr);
EXIT(GetRsrc);
END;
fFlag := kRsrcHdlValid;
fResAttrs := GetResAttrs(fHd1);
IF (ResError <> NoErr) THEN
fResAttrs := -1;
IF (fHd1^ = NIL) THEN
BEGIN
LoadResource(fHd1);
fLoaded := eWeLoadedIt;
IF gOption[eTrace] THEN
Trace('We loaded it');
END
ELSE
BEGIN
fLoaded := eAlreadyLoaded;
IF gOption[eTrace] THEN
Trace('Already loaded');
END;
IF (fHd1^ = NIL) THEN
BEGIN
gError := ResError;
IF (gError <> NoErr) THEN
BEGIN
ErrorMsg('Couldn't load resource',0);
IF (gError = memFullErr) THEN
ErrorMsg('No room in heap zone',1)
ELSE
ErrorOSerr('');
CommentWhich;
ReleaseRsrc(pRsrcPtr);
EXIT(GetRsrc);
END;
END;
fSize := SizeResource(fHd1);
END
ELSE
BEGIN
fFlag := kRsrcHdlValid;
fSize := MaxSizeRsrc(fHd1);

```

```

fLoaded := eNotYet;
IF gOption[eTrace] THEN
  Trace('No-load get, loaded not yet');
END;
IF (pIntIs = Index) THEN
  BEGIN
    GetResInfo(fHdl, fResId, fResType, sName);
    gError := ResError;
    IF (gError <> NoErr) THEN
      BEGIN
        ErrorOSErr('Couldn't get resource id');
        CommentWhich;
        ReleaseRsrc(pRsrcPtr);
        EXIT(GetRsrc);
      END;
    END;
  END;
  IF sResLoad THEN
    BEGIN
      fState := HGetState(fHdl);
      IF ((fResType = 'CODE')
        AND (fResId = 0)) THEN
        BEGIN
          MoveHHI(fHdl);
          HLock (fHdl);
        END
      ELSE
        HNoPurge(fHdl);
      END;
    END;
  END;
  IF gOption[eTrace] THEN
    TraceRsrc('Got', pRsrcPtr);
  INC(gCounts.fResources);
  INC(gTotals.fResources);
END;
PROCEDURE InitGlobals;
VAR
  sGetHdl, sPutHdl: DialogTHndl;
  sGetSize, sPutSize, sScrSize: Point;
BEGIN
  ZeroOutRange(@gAGlobals, @gZZGlobals);
  gCurrIOBuffer := NewPtr(kIOBufferSize);
  InitRsrc(@gCode0);
  InitRsrc(@gCurrRsrc);
  gEvtMask :=
    everyEvent - (updateMask + activMask);
  GetPort(gGrafPtr);
  gInd := ' ';
  sGetHdl := DialogTHndl(GetResource('DLOG', getDlgID));
  IF (sGetHdl = NIL)
  OR (LONGINT(sGetHdl) = -1) THEN
    SetPt(sGetSize, 304, 104)
  ELSE
    BEGIN
      IF (sGetHdl = NIL) THEN
        LoadResource(Handle(sGetHdl));
      sGetSize := sGetHdl^.boundsRect.botRight;
      ReleaseResource(Handle(sGetHdl));
    END;
  sPutHdl := DialogTHndl(GetResource('DLOG', putDlgID));
  IF (sPutHdl = NIL)
  OR (LONGINT(sPutHdl) = -1) THEN
    SetPt(sPutSize, 348, 136)
  ELSE
    BEGIN
      IF (sPutHdl = NIL) THEN
        LoadResource(Handle(sPutHdl));
      sPutSize := sPutHdl^.boundsRect.botRight;
      ReleaseResource(Handle(sPutHdl));
    END;
  END;
  WITH gGrafPtr^.portBits.bounds DO
    BEGIN
      sScrSize.h := right-left;
      sScrSize.v := bottom-top;
    END;
  END;

```

```

gSFGetPt.h := (sScrSize.h-sGetSize.h) DIV 2;
gSFGetPt.v := (sScrSize.v-sGetSize.v) DIV 2;
gSFPutPt.h := (sScrSize.h-sPutSize.h) DIV 2;
gSFPutPt.v := (sScrSize.v-sPutSize.v) DIV 2;
END;
PROCEDURE InitRsrc (pRsrcPtr: TRsrcPtr);
BEGIN
  ZeroOut(Ptr(pRsrcPtr), SIZEOF(TRsrcRec));
  pRsrcPtr^.fFlag := kRsrcIsInitd;
END;
FUNCTION JTEIsValid (pJTEPtr: TJTEPtr): BOOLEAN;
VAR
  sCode: TRsrcRec;
BEGIN
  IF gOption[eTrace] THEN
    Trace('JTEIsValid');
  JTEIsValid := FALSE;
  WITH pJTEPtr^, sCode DO
    BEGIN
      InitRsrc(@sCode);
      SetResLoad(FALSE);
      GetRsrc(@sCode, 'CODE', fSegId, ResId);
      SetResLoad(TRUE);
      IF (fFlag <> kRsrcHdlValid) THEN
        EXIT(JTEIsValid);
      JTEIsValid :=
        (fSkip3F3C = $3F3C) AND
        (fSegId > 0) AND
        (fSkipA9F0 = -22032) AND ( $A9F0 )
        (fSize > 0);
      ReleaseRsrc(@sCode);
    END;
  END;
END;
PROCEDURE ListCounts(pPtr: TCountsPtr);
BEGIN
  IF gOption[eTrace] THEN
    Trace('CountsListing');
  WITH pPtr^ DO
    BEGIN
      Writeln ('Files:');
      WritelnNbr(ffFiles, 6);
      Writeln (' processed');
      WritelnNbr(fExamined, 6);
      Writeln (' examined');
      WritelnNbr(fDeleted, 6);
      Writeln (' deleted');
      Writeln ('Resources:');
      WritelnNbr(fResources, 6);
      Writeln (' processed');
      WritelnNbr(fInfected, 6);
      Writeln (' infected');
      WritelnNbr(fRemoved, 6);
      Writeln (' removed');
      Writeln ('Currently available memory is ');
      WritelnNbr(MemAvail DIV 1024, 1);
      Writeln ('K. ');
      PauseBriefly;
    END;
  END;
END;
PROCEDURE LookForKnownViruses;
VAR
  sWeUsedToBeInfected: BOOLEAN;
PROCEDURE Get1stCode;
BEGIN
  WITH TJTHdl(gCode0.fHdl)^^.fJTEEntry[1] DO
    BEGIN
      GetRsrc(@gCurrRsrc, 'CODE', fSegId, ResId);
      IF (gCurrRsrc.fFlag <> kRsrcHdlValid) THEN
        BEGIN
          ErrorInfected('Couldn't get 1st CODE');
          InitRsrc(@gCurrRsrc);
          EXIT(LookForKnownViruses);
        END;
      END;
    END;
  END;
END;

```



```

END;
BEGIN
IF gOption[eTrace] THEN
  Trace('LookForKnownViruses');
sWeUsedToBeInfected := FALSE;
Get1stCode;
WITH gCurrRsrc DO
  BEGIN
    LookForVirus_nVIR;
    IF fInfected THEN
      BEGIN
        CountInfected;
        IF fKnown AND (fSize = 372) THEN
          ErrorInfected('nVIR 372 virus');
        ELSE IF fKnown AND (fSize = 422) THEN
          ErrorInfected('nVIR 422 virus');
        ELSE
          BEGIN
            ErrorInfected('New nVIR virus!');
            gFgPrTitle := '';
            CommentFgPrRsrc(@gCurrRsrc);
          END;
          IF Disinfected_nVIR THEN
            sWeUsedToBeInfected := TRUE;
          Get1stCode;
        END;
      LookForVirus_Scores;
      IF fKnown AND fInfected THEN
        BEGIN
          CountInfected;
          ErrorInfected('Scores virus');
          IF Disinfected_Scores THEN
            sWeUsedToBeInfected := TRUE;
          END
        ELSE
          ReleaseRsrc(@gCurrRsrc);
        END;
      IF sWeUsedToBeInfected THEN
        LookForKnownViruses;
      END;
    PROCEDURE PatrolBegins;
    BEGIN
      IF gOption[eTrace] THEN
        Trace('PatrolBegins');
      Writeln;
      Writeln('*****');
      ZeroOut(@gCounts, SIZEOF(TCountsRec));
      GetDateTime(gSecsBegins);
    END;
    PROCEDURE PatrolEnds;
    VAR
      sMins, sSecs: INTEGER;
    BEGIN
      GetDateTime(gSecsEnds);
      sSecs := gSecsEnds - gSecsBegins;
      sMins := sSecs DIV 60;
      sSecs := sSecs - (sMins * 60);
      Writeln;
      Writeln('*****');
      Writeln;
      Writeln('End of patrol that took ');
      Writeln(sMins, 1);
      WritelnChar(':');
      IF (sSecs < 10) THEN
        BEGIN
          WritelnChar('0');
          Writeln(sSecs, 1);
        END
      ELSE
        Writeln(sSecs, 2);
      Writeln;
      ListCounts(@gCounts);
    END;
  PROCEDURE PauseBriefly;

```

```

VAR
  sTicks: LONGINT;
BEGIN
  Delay(120, sTicks);
END;
PROCEDURE ProcessCodes;
VAR
  i, sNbrEntries, sPrevId: INTEGER;
  sWeirdCode0: BOOLEAN;
  PROCEDURE CommentWhere;
  BEGIN
    CommentBegins;
    Writeln('At entry ');
    Writeln(i, 1);
    Writeln;
  END;
BEGIN
  IF gOption[eTrace] THEN
    Trace('ProcessCodes');
  GetRsrc(@gCode0, 'CODE', 0, ResId);
  IF (gCode0.fFlag <> kRsrcHdlValid) THEN
    BEGIN
      ErrorMessage('Code rsrcs without CODE 0', 1);
      EXIT(ProcessCodes);
    END;
  IF NOT(Code0IsValid) THEN
    BEGIN
      ErrorMessage('Unexpected CODE 0 values', 1);
      ReleaseRsrc(@gCode0);
      EXIT(ProcessCodes);
    END;
  LookForKnownViruses;
  WITH TJTHdl(gCode0.fHdl) DO
    BEGIN
      sNbrEntries := fNbrBytesInTable DIV 8;
      sPrevId := 1;
      sWeirdCode0 :=
        (COPY(gCurrFilename, 1, 9) = 'Red Ryder') OR
        (COPY(gCurrFilename, 1, 6) = 'Canvas' ) OR
        (COPY(gCurrFilename, 1, 9) = 'PageMaker');
      FOR i := 1 TO sNbrEntries DO
        WITH fJEntry[i] DO
          BEGIN
            IF (fSkip3F3C = $3F3C)
              AND (fSegId = sPrevId)
              AND (fSkipA9F0 = -22032) THEN
              CYCLE;
            AbortPatrolIfCmdPeriodPressed;
            IF gAbortPatrol THEN
              LEAVE;
            IF NOT(JTEIsValid(@fJEntry[i])) THEN
              BEGIN
                ErrorMessage('CODE 0 has invalid JTE', 1);
                CommentWhere;
                LEAVE;
              END;
            IF sWeirdCode0 THEN
              BEGIN
                sPrevId := fSegId;
                CYCLE;
              END;
            IF (fSegId < sPrevId) THEN
              BEGIN
                ErrorMessage('JT not ascending', 1);
                CommentWhere;
                LEAVE;
              END;
            INC(sPrevId);
            IF (fSegId = sPrevId) THEN
              CYCLE;
            ErrorMessage('JT skips ResId', 1);
            CommentWhere;
            LEAVE;
          END;
        END;
      END;

```

```

END;
ReleaseRsrc(@gCode0);
END;
PROCEDURE ProcessFile;
VAR
  sSaveCIT: INTEGER;
PROCEDURE ExitIfCantReadFork;
BEGIN
  IF (gError <> NoErr) THEN
    BEGIN
      IF (gError = eofErr) THEN
        ( no resource fork )
      ELSE IF (gError = fnfErr) THEN
        ErrorMessage('File not found',1)
      ELSE IF (gError = nsvErr) THEN
        ErrorMessage('No such volume',1)
      ELSE IF (gError = opWrErr) THEN
        ErrorMessage(CONCAT('Already in use. ',
          'Don't use under MultiFinder!'),1)
      ELSE
        ErrorMessage('Couldn't open file');
      gError := NoErr;
      EXIT(ProcessFile);
    END;
  END;
BEGIN
  IF gOption[eTrace] THEN
    Trace('ProcessFile');
  AbortPatrolIfCmdPeriodPressed;
  IF gAbortPatrol THEN
    EXIT(ProcessFile);
  INC(gCounts.fFiles);
  INC(gTotals.fFiles);
  gInfected := FALSE;
  gInfectedWritten := FALSE;
  gReportFlags.fWroteFilename := FALSE;
  gScreenFlags.fWroteFilename := FALSE;
  IF gOption[eLList] THEN
    WryteFilename
  ELSE
    WryteFilenameToScreenOnlyForNow;
  IF (LENGTH(gCurrFilename) > 0) THEN
    IF (gCurrFilename[1] = '.') THEN
      BEGIN
        ErrorMessage('Filename begins with "."',1);
        EXIT(ProcessFile);
      END;
    IF gActiveSelf AND NOT(kProcessSelf) THEN
      EXIT(ProcessFile);
    gCurrEOF := -1;
    gError := FSOpen(gCurrFilename,gCurrWDRNum, gCurrRefNum);
    ExitIfCantReadFork;
    gError := GetEOF(gCurrRefNum,gCurrEOF);
    IF (gError = NoErr) THEN
      BEGIN
        WITH gCurrFInfo DO
          IF (COPY(gCurrFilename,1,7)='MacsBug')
            OR (fdType = 'RELB')
            OR (fdType = 'OBJ') THEN
            IF NOT(KnownDataFork) THEN
              CommentFgPrData;
            gError := FSClose(gCurrRefNum);
            IF (gError <> NoErr) THEN
              ErrorMessage('Couldn't close data fork');
            END
          ELSE
            ErrorMessage('Couldn't GetEOF');
        IF gActiveSelf THEN
          BEGIN
            gCurrRefNum := TWordPtr(kCurApRefNum);
            gError := NoErr;
          END
        ELSE IF gActiveSys THEN
          BEGIN

```

```

gCurrRefNum := TWordPtr(kSysMap);
gError := NoErr;
END
ELSE
  BEGIN
    SetResLoad(FALSE);
    gCurrRefNum := OpenRFPPerm(gCurrFilename,gCurrWDRNum,
      fsRdWrPerm);
    gError := ResError;
    SetResLoad(TRUE);
    ExitIfCantReadFork;
  END;
  IF (gCurrRefNum <> CurResFile) THEN
    BEGIN
      UseResFile(gCurrRefNum);
      gError := ResError;
      IF (gError <> NoErr) THEN
        BEGIN
          ErrorMessage('Couldn't use resource fork');
          gError := NoErr;
          EXIT(ProcessFile);
        END;
      END;
      INC(gCounts.fExamined);
      INC(gTotals.fExamined);
      IF (Count1Resources('CODE') > 0) THEN
        ProcessCodes;
      gFgPrTitle := 'Unknown Resource(s)';
      ProcessRsrcs('ADBS',@Process_ADBS);
      ProcessRsrcs('CACH',@Process_CACH);
      ProcessRsrcs('CDEF',@Process_CDEF);
      (etc)
      IF gOption[eFgPr] THEN
        BEGIN
          gFgPrTitle := 'Fingerprint(s)';
          ProcessRsrcs('ADBS',@ProcessCurrRsrc);
          ProcessRsrcs('CACH',@ProcessCurrRsrc);
          ProcessRsrcs('CDEF',@ProcessCurrRsrc);
          IF gOption[eFgPrC] THEN
            ProcessRsrcs('CODE',@ProcessCurrRsrc);
            ProcessRsrcs('DATA',@ProcessCurrRsrc);
            (etc)
            ProcessRsrcs('nVIR',@ProcessCurrRsrc);
          END;
        IF gActiveSelf OR gActiveSys THEN
          EXIT(ProcessFile);
        sSaveCIT := Count1Types;
        CloseResFile(gCurrRefNum);
        IF NOT(gInfected) THEN
          EXIT(ProcessFile);
        WITH gCurrFInfo DO
          BEGIN
            IF ((gCurrFilename = 'Note Pad File')
              OR (gCurrFilename = 'Scrapbook File'))
              AND (fdCreator = 'ZSYS')
              AND gOption[eRmVir] THEN
              BEGIN
                fdType := 'ZSYS';
                fdCreator := 'MACS';
                fdFlags := 4096;
                gError := SetFInfo(gCurrFilename,
                  gCurrWDRNum,
                  gCurrFInfo);
                IF (gError = NoErr) THEN
                  ErrorMessage('Reset to system document',0)
                ELSE
                  ErrorMessage('FInfo not reset');
                EXIT(ProcessFile);
              END;
            END;
          END;
        IF (gCurrEOF <> 0) THEN
          BEGIN
            ErrorMessage('File still has data fork',0);
            ErrorMessage('File not deleted',1);

```

```

EXIT(ProcessFile);
END;
IF (sSaveC1T <> 0) THEN
BEGIN
  ErrorMsg('File still has resources',0);
  ErrorMsg('File not deleted',1);
  EXIT(ProcessFile);
END;
ErrorMsg('File emptied',0);
gError := FSDelete(gCurrFilename,gCurrWDRNum);
IF (gError = NoErr) THEN
BEGIN
  gCurrFileDeleted := TRUE;
  INC(gCounts.fDeleted);
  INC(gTotals.fDeleted);
  ErrorMsg('File deleted',1);
END
ELSE
  Error0SErr('File not deleted');
END;
PROCEDURE ProcessRsrcs(pResType:ResType; pProcPtr: ProcPtr);
VAR
  i,sIdx: INTEGER;
BEGIN
  IF gOption[eTrace] THEN
    Trace('ProcessRsrcs');
  WITH gCurrRsrc DO
  BEGIN
    sIdx := 1;
    FOR i := 1 TO Count1Resources(pResType) DO
    BEGIN
      AbortPatrolIfCmdPeriodPressed;
      IF gAbortPatrol THEN
        LEAVE;
      GetRsrc(@gCurrRsrc,pResType,sIdx,Index);
      IF (fFlag <> kRsrcHdlValid) THEN
        BEGIN
          INC(sIdx);
          CYCLE;
        END;
      CallProcPtr(pProcPtr);
      IF fInfected THEN
        BEGIN
          Count1Infected;
          ErrorInfected('');
          CommentRsrcBegins(@gCurrRsrc);
          Writeln(' is an infection');
          IF RemovedRsrc(@gCurrRsrc) THEN
            BEGIN
              ErrorMsg('Removed',0);
              CYCLE;
            END;
          ErrorMsg('Not removed',1);
          INC(sIdx);
          CYCLE;
        END;
      IF NOT(fKnown) THEN
        CommentFgPrRsrc(@gCurrRsrc);
      ReleaseRsrc(@gCurrRsrc);
      INC(sIdx);
    END;
  END;
END;
PROCEDURE ReleaseRsrc(pRsrcPtr: TRsrcPtr);
BEGIN
  WITH pRsrcPtr^ DO
  BEGIN
    IF (fFlag <> kRsrcHdlValid) THEN
    BEGIN
      ErrorMsg('Error using ReleaseRsrc',4);
      AwaitKeypress;
      ExitSecurityPatrol;
    END;
    IF gOption[eTrace] THEN

```

```

      TraceRsrc('About to release',pRsrcPtr);
      IF (gActiveSelf OR gActiveSys) THEN
        IF gOption[eTrace] THEN
          Trace('Not Released')
        ELSE
          BEGIN
            HSetState(fHdl,fState);
            ReleaseResource(fHdl);
            IF gOption[eTrace] THEN
              Trace('Released');
            END;
            InitRsrc(pRsrcPtr);
          END;
        END;
      FUNCTION RemovedRsrc(pRsrcPtr: TRsrcPtr): BOOLEAN;
      VAR
        sBits0and7:LONGINT;
      PROCEDURE ExitIfError(pStr: Str255);
      BEGIN
        gError := ResError;
        IF (gError <> NoErr) THEN
          BEGIN
            ErrorMsg(pStr,0);
            IF (gError = wPrErr) THEN
              ErrorMsg('Disk is locked',0)
            ELSE
              Error0SErr('');
            CommentRsrcBegins(pRsrcPtr);
            Writeln(' not removed');
            ReleaseRsrc(pRsrcPtr);
            EXIT(RemovedRsrc);
          END;
        END;
      BEGIN
        RemovedRsrc := FALSE;
        IF gOption[eTrace] THEN
          Trace('RemovedRsrc');
        AbortPatrolIfCmdPeriodPressed;
        IF gAbortPatrol
        OR NOT(gOption[eRmVir]) THEN
          BEGIN
            ReleaseRsrc(pRsrcPtr);
            EXIT(RemovedRsrc);
          END;
        WITH pRsrcPtr^ DO
        BEGIN
          IF (fFlag <> kRsrcHdlValid) THEN
            BEGIN
              ErrorMsg('Error using RemovedRsrc',4);
              AwaitKeypress;
              ExitSecurityPatrol;
            END;
            IF gOption[eTrace] THEN
              BEGIN
                TraceRsrc('About to remove',pRsrcPtr);
                AbortPatrolIfCmdPeriodPressed;
                IF gAbortPatrol THEN
                  EXIT(RemovedRsrc);
                END;
              IF NOT(fInfected) THEN
                BEGIN
                  ErrorMsg('Tried to remove uninfected',4);
                  AwaitKeypress;
                  ExitSecurityPatrol;
                END;
              IF kZeroOutVirs AND (fHdl^ <> NIL) THEN
                BEGIN
                  ZeroOut(fHdl^,fSize);
                  ChangedResource(fHdl);
                  gError := ResError;
                  IF (gError = NoErr) THEN
                    BEGIN
                      WriteResource(fHdl);

```

```

gError := ResError;
IF (gError <> NoErr) THEN
  Error0SErr('Couldn't WriteResource');
END
ELSE
  Error0SErr('Couldn't ChangedResource');
END;
sBits0and7 := BAnd(fResAttrs,$81);
SetResAttrs(fHdl,LoWord(sBits0and7));
RmveResource(fHdl);
ExitIfError('Couldn't remove resource');
UpdateResFile(gCurrRefNum);
ExitIfError('Couldn't update res file');
DisposHandle(fHdl);
InitRsrc(pRsrcPtr);
RemovedRsrc := TRUE;
IF gOption[Trace] THEN
  Trace('RemovedRsrc successful');
END;
INC(gCounts.fRemoved);
INC(gTotals.fRemoved);
END;
PROCEDURE ShortHexDump
(pPtr: Ptr;
pNbrBytes: SignedByte);
VAR
i: INTEGER;
sCh1,sCh2,sDigit: LONGINT;
sIdx: Ptr;
BEGIN
sIdx := pPtr;
FOR i := 1 TO pNbrBytes DO
  BEGIN
    sDigit := ORD4(sIdx^);
    sCh1 := BSR(BAnd(sDigit,$F0),4);
    sCh2 := BAnd(sDigit,$0F);
    IF sCh1 > 9 THEN
      Writeln(CHR(sCh1 + $37))
    ELSE
      Writeln(CHR(sCh1 + $30));
    IF sCh2 > 9 THEN
      Writeln(CHR(sCh2 + $37))
    ELSE
      Writeln(CHR(sCh2 + $30));
    INC(LONGINT(sIdx));
  END;
END;
PROCEDURE Trace(pStr: Str255);
BEGIN
ErrorBegins(pStr);
ErrorEnds(0);
END;
PROCEDURE TraceNbr(pStr: Str255;pNbr: LONGINT);
BEGIN
ErrorBegins(pStr);
WritelnNbr(pNbr,1);
ErrorEnds(0);
END;
PROCEDURE TraceRsrc(pStr: Str255;pRsrcPtr: TRsrcPtr);
BEGIN
ErrorBegins(pStr);
WITH pRsrcPtr^ DO
  BEGIN
    Writeln(' ');
    WritelnType(fResType);
    WritelnNbr(fResId,7);
  END;
ErrorEnds(0);
END;
PROCEDURE ZeroOut(pStart: Ptr;pCount: Size);
VAR
i: INTEGER;
sIdx: Ptr;
BEGIN

```

```

sIdx := pStart;
FOR i := 1 TO pCount DO
  BEGIN
    sIdx^ := 0;
    INC(LONGINT(sIdx));
  END;
END;
PROCEDURE ZeroOutRange(p1: Ptr; p2: Ptr);
VAR
i: INTEGER;
sIdx: Ptr;
BEGIN
IF (ORD4(p1) < ORD4(p2)) THEN
  sIdx := p1
ELSE
  sIdx := p2;
FOR i := 1 TO ABS(ORD4(p2)-ORD4(p1))+1 DO
  BEGIN
    sIdx^ := 0;
    INC(LONGINT(sIdx));
  END;
END;
END.

```

"MainDlog.p"

```

UNIT MainDlog;
INTERFACE
USES
  MemTypes,QuickDraw,OSIntf,ToolIntf, PackIntf,Globals;
PROCEDURE InitMainDlog;
FUNCTION MainDlogWorkRequested: TMainItem;
IMPLEMENTATION
($R-)
CONST
  (---item range---)
  kItemFst      = eDirs;
  kItemLst      = eDBtn;
  (---item type subranges---)
  kBtnFst       = eDirs;
  kBtnLst       = eQuit;
  kChkFst       = eAwait;
  kChkLst       = eTrace;
  kStatFst      = eMain;
  kStatLst      = eScOW;
  kUIItemFst    = eDBtn;
  kUIItemLst    = eDBtn;
  (---titled item subrange---)
  kTIItemFst    = eDirs;
  kTIItemLst    = eScOW;
TYPE
  TDitmPtr      = ^TDitmRec;
  TDitmRec      =
    PACKED RECORD
      fProcPtr:  ProcPtr;
      fRect:     Rect;
      fType:     Byte;
      fLen:      Byte;
      fData:     INTEGER;
    END;
VAR
  gDBtnRect:    Rect;
  gHdl:         ARRAY [eAwait..eTrace]
                OF ControlHandle;
  gRect:        ARRAY [eAwait..eTrace]
                OF Rect;
PROCEDURE ChkChk(pItem: TMainItem);
BEGIN
IF (pItem < kChkFst)
OR (pItem > kChkLst) THEN
  BEGIN
    SysBeep(3);
    EXIT(ChkChk);
  END;
SetCtlValue(gHdl[pItem],ORD(gOption[pItem]));
IF gDisabled[pItem] THEN

```

```

BEGIN
  SetDItem(gDlogPtr,ORD(pitem),
    ctrlItem+chkCtrl+itemDisable,
    Handle(gHdl[pitem]),gRect[pitem]);
  HiliteControl(gHdl[pitem],255);
END
ELSE
BEGIN
  SetDItem(gDlogPtr,ORD(pitem),
    ctrlItem+chkCtrl,
    Handle(gHdl[pitem]),gRect[pitem]);
  HiliteControl(gHdl[pitem],0);
END;
END;
PROCEDURE FrameDefaultBtn(pWindowPtr:WindowPtr; pitemNo:
INTEGER);
VAR
  sPenState: PenState;
BEGIN
  GetPenState(sPenState);
  PenSize (3,3);
  FrameRoundRect(gDBtnRect,16,16);
  SetPenState(sPenState);
END;
PROCEDURE InitMainDialog;
CONST
  kTitleMax      = 27;
  kTitmLen       = 42;
  ( 14 + kTitleMax + ord(odd(kTitleMax)); )
  kUITmLen       = 14;
VAR
  i:              TMainItem;
  sDitmPtr:       TDitmPtr;
  sDlogRect:      Rect;
  sHdl:           Handle;
  sNbrTlms:       INTEGER;
  sNbrUtlms:      INTEGER;
  sRect:          ARRAY [eDirs..eScOW]
                  OF Rect;
  sSize:          Size;
  sTitle:         ARRAY [eDirs..eScOW]
                  OF STRING[kTitleMax];
  sType:          INTEGER;
BEGIN
  IF gOption[eTrace] THEN
    Trace('InitMainDialog');
  FOR i := kTitmFst TO kTitmLst DO
    IF (i >= kBtnFst)
      AND (i <= kBtnLst) THEN
      BEGIN
        SetRect (sRect[i], 266, 65, 346, 83);
        OffsetRect(sRect[i],0, 27*(ORD(i)-ORD(kBtnFst)));
      END
    ELSE IF (i >= kChkFst)
      AND (i <= kChkLst) THEN
      BEGIN
        SetRect (sRect[i], 24, 62, 185, 80);
        OffsetRect(sRect[i],0, 20*(ORD(i)-ORD(kChkFst)));
      END
    ELSE
      SetRect (sRect[i], 12, 42, 216, 60);
  OffsetRect(sRect[eMain], 080,-30);
  OffsetRect(sRect[eOpts], 000,000);
  OffsetRect(sRect[eScOW], 216,000);
  gDBtnRect := sRect[kItemFst];
  InsetRect (gDBtnRect, -4, -4);
  WITH sDlogRect, gSFGetPt DO
    BEGIN
      top    := v - 10;
      left   := h - 10;
      bottom := top + 210;
      right  := left + 368;
    END;
  sTitle[eDirs] := 'Directories';

```

```

  sTitle[eDiry] := 'Directory';
  sTitle[eEvery] := 'Everything';
  sTitle[eFiles] := 'Files';
  sTitle[eQuit] := 'Quit';
  sTitle[eAwait] := 'Await Keypress';
  sTitle[eBeeps] := 'Beep';
  sTitle[eFgPr] := 'Fingerprint';
  sTitle[eFgPrC] := 'Fingerprint CODEs';
  sTitle[eLList] := 'Long Listing';
  sTitle[eRmVir] := 'Remove Viruses';
  sTitle[eTrace] := 'Trace';
  sTitle[eMain] :=
    'Security Patrol Main Dialog';
  sTitle[eOpts] := 'Options';
  sTitle[eScOW] := 'Scope Of Work';
  sNbrTlms := (ORD(kTitmLst)-ORD(kTitmFst))+1;
  sNbrUtlms := (ORD(kUITmLst)-ORD(kUITmFst))+1;
  sHdl := NewHandle(2 + (sNbrTlms*kTitmLen) +
    (sNbrUtlms*kUITmLen));
  TWordPtr(sHdl)^ := ORD(kItemLst) - 1;
  sSize := 2;
  FOR i := kItemFst TO kItemLst DO
    BEGIN
      IF gOption[eTrace] THEN
        TraceNbr('sSize = ',sSize);
      sDitmPtr := POINTER(ORD4(sHdl^)+sSize);
      WITH sDitmPtr^ DO
        IF (i >= kTitmFst)
          AND (i <= kTitmLst) THEN
          BEGIN
            fProcPtr := NIL;
            fRect := sRect[i];
            IF (i <= kBtnLst) THEN
              fType := ctrlItem + btnCtrl
            ELSE IF (i <= kChkLst) THEN
              fType := ctrlItem + chkCtrl
            ELSE
              fType := statText + itemDisable;
            BlockMove(@sTitle[i],@fLen,
              LENGTH(sTitle[i])+1);
            sSize := sSize+14+fLen+ORD(ODD(fLen));
          END
        ELSE IF (i = eDBtn) THEN
          BEGIN
            fProcPtr := @FrameDefaultBtn;
            fRect := gDBtnRect;
            fType := userItem + itemDisable;
            fLen := 0;
            sSize := sSize + 14;
          END
        ELSE
          SysBeep(60);
      END;
    SetHandleSize(sHdl,sSize);
    gDlogPtr := NewDialog(NIL,sDlogRect,'',
      FALSE,dBoxProc,POINTER(-1),FALSE,0,sHdl);
    FOR i := kChkFst TO kChkLst DO
      BEGIN
        GetDItem(gDlogPtr,ORD(i),sType,
          Handle(gHdl[i]),gRect[i]);
        IF gOption[eTrace] THEN
          TraceNbr('ChkChk'ing item ',ORD(i));
        ChkChk(i);
      END;
    END;
  END;
FUNCTION KybdEquivsFilter(pDialogPtr: DialogPtr;
VAR pEventRec: EventRecord;VAR pitemHit: INTEGER): BOOLEAN;
VAR
  sChar:
  RECORD
    CASE INTEGER OF
      0:(F1,F2,F3: SignedByte;
        Enum: TMainItem);

```

```

1:(L:      LONGINT);
END;
sItem:      TMainItem;
BEGIN
sItem := eNotADLogItem;
WITH pEventRec DO
  IF (what = keyDown) THEN
    BEGIN
      sChar.L := BAnd(message, charCodeMask);
      IF (sChar.L = $03)
      OR (sChar.L = $0D) THEN
        sItem := eDirs
      ELSE IF (sChar.L = ORD('D'))
      OR (sChar.L = ORD('d')) THEN
        sItem := eDiry
      ELSE IF (sChar.L = ORD('E'))
      OR (sChar.L = ORD('e')) THEN
        sItem := eEvery
      ELSE IF (sChar.L = ORD('F'))
      OR (sChar.L = ORD('f')) THEN
        sItem := eFiles
      ELSE IF (sChar.L = ORD('Q'))
      OR (sChar.L = ORD('q'))
      OR ((BAnd(modifiers, cmdKey) <> 0)
      AND (sChar.L = ORD('.'))) THEN
        sItem := eQuit
      ELSE
        BEGIN
          sChar.L :=
            (sChar.L - ORD4('1')) + ORD(kChkFst);
          IF (sChar.L >= ORD(kChkFst))
          AND (sChar.L <= ORD(kChkLst)) THEN
            IF NOT(gDisabled[sChar.Enum]) THEN
              sItem := sChar.Enum;
            END;
          END;
        END;
      IF (sItem = eNotADLogItem) THEN
        KybdEquivsFilter := FALSE
      ELSE
        BEGIN
          pItemHit := ORD(sItem);
          KybdEquivsFilter := TRUE;
        END;
      END;
    END;
  FUNCTION MainDlogWorkRequested      TMainItem;
  VAR
    sItem:
      RECORD
        CASE INTEGER OF
          0:(Filler: SignedByte;
            Enum: TMainItem);
          1:(Int: INTEGER);
        END;
    BEGIN
      IF gOption[eTrace] THEN
        Trace('MainDlogWorkRequested');
      BringToFront(gDlogPtr);
      ShowWindow (gDlogPtr);
      ModalDialog (@KybdEquivsFilter, sItem.Int);
      WHILE (sItem.Enum >= kChkFst)
      AND (sItem.Enum <= kChkLst) DO
        BEGIN
          gOption[sItem.Enum] :=
            NOT(gOption[sItem.Enum]);
          ChkChk(sItem.Enum);
          IF (sItem.Enum = eFgPr) THEN
            BEGIN
              gDisabled[eFgPrC] := NOT(gOption[eFgPr]);
              gOption [eFgPrC] := FALSE;
              ChkChk(eFgPrC);
            END;
          ModalDialog (@KybdEquivsFilter, sItem.Int);
        END;
      HideWindow (gDlogPtr);

```

```

SetPort (gGrafPtr);
IF gOption[eTrace] THEN
  TraceNbr('Returning ', ORD(sItem.Enum));
IF (sItem.Enum >= kItemFst)
AND (sItem.Enum <= kItemLst) THEN
  MainDlogWorkRequested := sItem.Enum
ELSE
  MainDlogWorkRequested := eQuit;
END;
END.

                                "Patrol.p"
UNIT Patrol;
INTERFACE
USES
  MemTypes, QuickDraw, OSIntf, ToolIntf, PackIntf, Globals;
PROCEDURE BuildDirname;
PROCEDURE InitPatrols;
PROCEDURE PatrolDirectories(pOnly1Deep: BOOLEAN);
PROCEDURE PatrolEverything;
PROCEDURE PatrolFiles;
IMPLEMENTATION
($R-)
CONST
  kPatsInitd      = -12345;
TYPE
  TOverlappingPBs =
    RECORD
      CASE INTEGER OF
        0: (fPbRec: HParamBlockRec);
        1: (fCPbRec: CInfoPbRec);
      END;
  VAR
    gAAPatImpl, gZZPatImpl: SignedByte;
    gAppDirId, gInitdFlag, gSysDirId: LONGINT;
    gAppVRefNum, gOrigWDRefNum, gSysVRefNum: INTEGER;
    gOnly1Deep: BOOLEAN;
    gPBs: TOverlappingPBs;
    gSFLst: SFTypelist;
    gWDPBRec: WDPBRec;
  PROCEDURE BuildDirname;
  VAR
    sErr: OSErr;
    sLen: INTEGER;
    sName: Str255;
    sPBs: TOverlappingPBs;
  BEGIN
    IF gOption[eTrace] THEN
      Trace('BuildDirname');
    WITH sPBs, fPbRec, fCPbRec DO
      BEGIN
        sPBs := gPBs;
        ioNamePtr := @sName;
        ioVRefNum := gCurrWDRefNum;
        gCurrDirname := '';
        IF gHFS THEN
          BEGIN
            ioDirIndex := -1;
            ioDrParID := 0;
            REPEAT
              ioDrDirId := ioDrParID;
              sErr := PBGetCatInfo(@fCPbRec, FALSE);
              IF (sErr <> NoErr) THEN
                EXIT(BuildDirname);
              sLen := LENGTH(sName) + 1 + LENGTH(gCurrDirname);
              IF (sLen <= 255) THEN
                gCurrDirname := CONCAT(sName, ': ', gCurrDirname);
              UNTIL ioDrDirId = 2;
            END
          ELSE
            BEGIN
              sErr := PBGetVol(@fPbRec, FALSE);
              IF (sErr = NoErr) THEN
                gCurrDirname := CONCAT(sName, ': ');
            END;
          END;
        END;

```

```

END;
END;
PROCEDURE CallProcessFile;
BEGIN
  gActiveSelf :=
    (gCurrVRefNum = gAppVRefNum) AND
    (gCurrDirId = gAppDirId) AND
    (gCurrFilename = StringPtr(kCurApName)^);
  gActiveSys :=
    (gCurrVRefNum = gSysVRefNum) AND
    (gCurrDirId = gSysDirId) AND
    (gCurrFilename = StringPtr(kSysResName)^);
  gCurrFileDeleted := FALSE;
  ProcessFile;
END;
PROCEDURE FloatWDDeeper(pDrDirId: LONGINT);
BEGIN
  IF gOption[eTrace] THEN
    TraceNbr('Begin FloatWDDeeper, WD = ',
      ORD4(gCurrWRefNum));
  WITH gPBs, fCPBRec DO
    BEGIN
      IF NOT((pDrDirId=0) OR (pDrDirId=2)) THEN
        BEGIN
          gWDPBRec.ioVRefNum := gCurrWRefNum;
          gWDPBRec.ioWDDirId := 0;
          gError := PBCloseWD(@gWDPBRec, FALSE);
          IF (gError <> NoErr) THEN
            BEGIN
              ErrorOSErr('Couldn't close WD');
              EXIT(FloatWDDeeper);
            END;
          END;
          gWDPBRec.ioVRefNum := gCurrVRefNum;
          gWDPBRec.ioWDDirId := ioDrDirId;
          gError := PBOpenWD(@gWDPBRec, FALSE);
          IF (gError <> NoErr) THEN
            BEGIN
              ErrorOSErr('Couldn't open subdir WD');
              EXIT(FloatWDDeeper);
            END;
          gCurrWRefNum := gWDPBRec.ioVRefNum;
          END;
        IF gOption[eTrace] THEN
          TraceNbr('End FloatWDDeeper, WD = ',
            ORD4(gCurrWRefNum));
        END;
      PROCEDURE FloatWDShallower(pDrDirId: LONGINT);
      BEGIN
        IF gOption[eTrace] THEN
          TraceNbr('Begin FloatWDShallower, WD = ',
            ORD4(gCurrWRefNum));
        WITH gPBs, fCPBRec DO
          BEGIN
            gError := PBCloseWD(@gWDPBRec, FALSE);
            gWDPBRec.ioVRefNum := gCurrWRefNum;
            gWDPBRec.ioWDDirId := 0;
            IF (gError <> NoErr) THEN
              BEGIN
                ErrorOSErr('Couldn't close subdir WD');
                EXIT(FloatWDShallower);
              END;
            IF (pDrDirId = 0)
            OR (pDrDirId = 2) THEN
              gCurrWRefNum := gOrigWRefNum
            ELSE
              BEGIN
                gWDPBRec.ioVRefNum := gCurrVRefNum;
                gWDPBRec.ioWDDirId := pDrDirId;
                gError := PBOpenWD(@gWDPBRec, FALSE);
                IF (gError <> NoErr) THEN
                  BEGIN
                    ErrorOSErr('Couldn't reopen WD');
                    EXIT(FloatWDShallower);

```

```

          END;
          gCurrWRefNum := gWDPBRec.ioVRefNum;
          END;
        END;
      IF gOption[eTrace] THEN
        TraceNbr('End FloatWDShallower, WD = ',
          ORD4(gCurrWRefNum));
      END;
    END;
  END;
  PROCEDURE GetActualDirId(pDrDirId: LONGINT);
  BEGIN
    WITH gPBs, fCPBRec DO
      BEGIN
        ioDrDirId := pDrDirId;
        ioFDirIndex := -1;
        ioVRefNum := gOrigWRefNum;
        gError := PBGetCatInfo(@fCPBRec, FALSE);
        IF (gError <> NoErr) THEN
          BEGIN
            ErrorOSErr('Couldn't GetActualDirId');
            EXIT(GetActualDirId);
          END;
        gCurrDInfo := ioDrUsrWds;
        gCurrDirId := ioDrDirId;
        IF gOption[eTrace] THEN
          TraceNbr('ActualDirID = ', gCurrDirId);
        END;
      END;
    END;
  END;
  PROCEDURE InitPatrols;
  BEGIN
    IF gOption[eTrace] THEN
      Trace('InitPatrols');
    WITH gPBs, fPBRec, fCPBRec DO
      BEGIN
        ZeroOutRange(@gAAPatImpl, @gZZPatImpl);
        ghFS := TWordPtr(kSFCBLen)^ > 0;
        ioNamePtr := @gCurrFilename;
        IF ghFS THEN
          BEGIN
            gError := GetVRefNum
              (TWordPtr(kSysMap)^, gSysVRefNum);
            IF (gError <> NoErr) THEN
              ErrorOSErr('Couldn't get act sys vol');
            ioVRefNum := gSysVRefNum;
            gError := PBHGetVInfo(@fPBRec, FALSE);
            IF (gError <> NoErr) THEN
              ErrorOSErr('Couldn't get act sys dir');
            IF (fPBRec.ioVfndrInfo[1] = 0) THEN
              ErrorOSErr('Boot vol not Blessed');
            gSysDirId := ioVfndrInfo[1];
            gError := PBHGetVol(@fPBRec, FALSE);
            IF (gError <> NoErr) THEN
              ErrorOSErr('Couldn't get own DirId');
            gAppDirId := ioDirId;
            gCurrDirId := ioDirId;
            gCurrWRefNum := ioVRefNum;
            ioVolIndex := 0;
            gError := PBHGetVInfo(@fPBRec, FALSE);
            IF (gError <> NoErr) THEN
              ErrorOSErr('Couldn't get own VInfo');
            gAppVRefNum := ioVRefNum;
            END;
          ELSE
            BEGIN
              gSysVRefNum := TWordPtr(kBootDrive)^;
              gError := PBGetVol(@fPBRec, FALSE);
              IF (gError <> NoErr) THEN
                ErrorOSErr('Couldn't get own Vol');
              gAppVRefNum := ioVRefNum;
              gCurrWRefNum := ioVRefNum;
              gAppDirId := 2;
              gSysDirId := 2;
              gCurrDirId := 2;
            END;
          BuildDirname;

```

```

gCurrFilename := StringPtr(kCurApName)^;
ioDirIndex := 0;
ioDirId := 0;
ioVRefNum := gCurrWRefNum;
gError := PBGetFInfo(@fPBRec,FALSE);
IF (gError <> NoErr) THEN
    ErrorOSErr('Couldn't get own FInfo');
gCurrFInfo := ioFInfo;
gActiveSelf := TRUE;
gActiveSys := FALSE;
gWDPBRec.ioWDProcID := $50617472; ('Patr')
gInitdFlag := kPatsInitd;
END;
END;
PROCEDURE PatrolDir(pDrDirId: LONGINT);
VAR
    sIndex: INTEGER;
BEGIN
    IF gOption[Trace] THEN
        TraceNbr('PatrolDir ',pDrDirId);
    WITH gPBs,fCPBRec DO
        BEGIN
            IF (gInitdFlag <> kPatsInitd) THEN
                BEGIN ( shouldn't happen )
                    ErrorOSErr('InitPatrols not done');
                    EXIT(PatrolDir);
                END;
            IF gHFS THEN
                BEGIN
                    gCurrDirId := pDrDirId;
                    GetActualDirId(pDrDirId);
                    IF (gError <> NoErr) THEN
                        EXIT(PatrolDir);
                    END
                END
            ELSE
                gCurrDirId := 2;
                BuildDirname;
                DirectoryBegins;
                sIndex := 1;
                REPEAT
                    gCurrIndex := sIndex;
                    ioDirIndex := sIndex;
                    ioDirId := 0;
                    ioVRefNum := gCurrWRefNum;
                    gError := PBGetFInfo(@fPBRec,FALSE);
                    IF (gError = NoErr) THEN
                        BEGIN
                            gCurrFInfo := ioFInfo;
                            CallProcessFile;
                        END
                    ELSE IF (gError <> fnfErr) THEN
                        BEGIN
                            ErrorOSErr('Couldn't get a file');
                            EXIT(PatrolDir);
                        END;
                    IF NOT(gCurrFileDeleted) THEN
                        INC(sIndex);
                UNTIL (gError <> NoErr) OR gAbortPatrol;
                IF gAbortPatrol THEN
                    EXIT(PatrolDir);
                IF (gError <> fnfErr) THEN
                    BEGIN
                        ErrorOSErr('Error at end of files');
                        EXIT(PatrolDir);
                    END;
                gError := NoErr;
                IF gHFS AND NOT(gOnly1Deep) THEN
                    BEGIN
                        sIndex := 1;
                        REPEAT
                            gCurrIndex := sIndex;
                            ioDirIndex := sIndex;
                            ioDirId := 0;
                            ioVRefNum := gCurrWRefNum;

```

```

gError := PBGetCatInfo(@fCPBRec,FALSE);
IF (gError = NoErr) THEN
    BEGIN
        IF BTst(ORD4(ioF1Attrib),4) THEN
            BEGIN
                FloatWDDeeper (pDrDirId);
                IF (gError <> NoErr) THEN
                    EXIT(PatrolDir);
                PatrolDir(ioDrDirId);
                IF (gError <> NoErr)
                    AND (pDrDirId <> 0)
                    AND (pDrDirId <> 2) THEN
                    EXIT(PatrolDir);
                FloatWDSlower(pDrDirId);
                IF (gError <> NoErr) THEN
                    EXIT(PatrolDir);
                END;
            END
        ELSE IF (gError <> fnfErr) THEN
            BEGIN
                ErrorOSErr('Couldn't get a dir');
                EXIT(PatrolDir);
            END;
            INC(sIndex);
            UNTIL (gError <> NoErr) OR gAbortPatrol;
            IF gAbortPatrol THEN
                EXIT(PatrolDir);
            IF (gError <> fnfErr) THEN
                BEGIN
                    ErrorOSErr('Error at end of subdirs');
                    EXIT(PatrolDir);
                END;
            gError := NoErr;
            gCurrDirId := pDrDirId;
            GetActualDirId(pDrDirId);
            IF (gError <> NoErr) THEN
                EXIT(PatrolDir);
            BuildDirname;
            END;
            DirectoryEnds;
        END;
    END;
PROCEDURE PatrolDirectories(pOnly1Deep:BOOLEAN);
BEGIN
    IF gOption[Trace] THEN
        Trace('PatrolDirectories ');
    WITH gPBs,fPBRec,fCPBRec,gSFRep DO
        BEGIN
            IF (gInitdFlag <> kPatsInitd) THEN
                BEGIN ( shouldn't happen )
                    ErrorOSErr('InitPatrols not done');
                    EXIT(PatrolDirectories);
                END;
            gOnly1Deep := pOnly1Deep;
            SGetFile
                (gSFGetPt,'',NIL,-1,gSFLst,NIL,gSFRep);
            WHILE good DO
                BEGIN
                    IF gHFS THEN
                        BEGIN
                            ioVRefNum := gSFRep.vRefNum;
                            ioVolIndex := 0;
                            gError := PBHGetVInfo(@fPBRec,FALSE);
                            IF (gError <> NoErr) THEN
                                BEGIN
                                    ErrorOSErr('Couldn't get own VInfo');
                                    LEAVE;
                                END;
                            gCurrVRefNum := ioVRefNum;
                        END
                    ELSE
                        gCurrVRefNum := vRefNum;
                        gCurrWRefNum := vRefNum;
                        gOrigWRefNum := vRefNum;

```



```

PatrolBegins;
PatrolDir(0);
PatrolEnds;
IF (gError <> NoErr) THEN
    LEAVE;
SFGetFile
    (gSFGetPt, '', NIL, -1, gSFLst, NIL, gSFRep);
END;
gError := NoErr;
END;
END;
PROCEDURE PatrolEverything;
VAR
    sIndex: INTEGER;
BEGIN
    IF gOption[Trace] THEN
        Trace('PatrolEverything ');
    WITH gPBs, fPBRec, fCPBRec DO
        BEGIN
            IF (gInitdFlag <> kPatsInitd) THEN
                BEGIN ( shouldn't happen )
                    ErrorOSErr('InitPatrols not done');
                    EXIT(PatrolEverything);
                END;
            gOnlyIDeep := FALSE;
            PatrolBegins;
            ioVRefNum := 0;
            sIndex := 1;
            REPEAT
                gCurrIndex := sIndex;
                ioVolIndex := sIndex;
                gError := PBGetVInfo(@fPBRec, FALSE);
                IF (gError = NoErr) THEN
                    BEGIN
                        gCurrVRefNum := ioVRefNum;
                        gCurrWRefNum := ioVRefNum;
                        gOrigWRefNum := ioVRefNum;
                        PatrolDir(2);
                        IF (gError <> NoErr) THEN
                            EXIT(PatrolEverything);
                        INC(sIndex);
                    END
                ELSE IF (gError <> nsvErr) THEN
                    BEGIN
                        ErrorOSErr('Couldn't get a volume');
                        EXIT(PatrolEverything);
                    END;
            UNTIL gError <> NoErr;
            IF (gError <> nsvErr) THEN
                BEGIN
                    ErrorOSErr('Error at end of volumes');
                    EXIT(PatrolEverything);
                END;
            gError := NoErr;
            PatrolEnds;
        END;
    END;
END;
PROCEDURE PatrolFiles;
VAR
    sPrevDirId: LONGINT;
    sPrevVRefNum: INTEGER;
    sPrevWRefNum: INTEGER;
    PROCEDURE CallPrevDirEnd;
    VAR
        sTempDirId: LONGINT;
        sTempVRefNum: INTEGER;
        sTempWRefNum: INTEGER;
    BEGIN
        IF (sPrevWRefNum <> 0) THEN
            BEGIN
                sTempDirId := gCurrDirId;
                sTempVRefNum := gCurrVRefNum;
                sTempWRefNum := gCurrWRefNum;
                gCurrDirId := sPrevDirId;

```

```

                gCurrVRefNum := sPrevVRefNum;
                gCurrWRefNum := sPrevWRefNum;
                DirectoryEnds;
                gCurrDirId := sTempDirId;
                gCurrVRefNum := sTempVRefNum;
                gCurrWRefNum := sTempWRefNum;
            END;
        END;
    BEGIN
        IF gOption[Trace] THEN
            Trace('PatrolFiles ');
        WITH gPBs, fPBRec, fCPBRec, gSFRep DO
            BEGIN
                IF (gInitdFlag <> kPatsInitd) THEN
                    BEGIN ( shouldn't happen )
                        ErrorOSErr('InitPatrols not done');
                        EXIT(PatrolFiles);
                    END;
                PatrolBegins;
                sPrevWRefNum := 0;
                SFGetFile
                    (gSFGetPt, '', NIL, -1, gSFLst, NIL, gSFRep);
                WHILE good DO
                    BEGIN
                        IF gHFS THEN
                            BEGIN
                                ioVRefNum := gSFRep.vRefNum;
                                ioDrDirId := 0;
                                ioFDirIndex := -1;
                                gError := PBGetCatInfo(@fPBRec, FALSE);
                                IF (gError <> NoErr) THEN
                                    BEGIN
                                        ErrorOSErr('Couldn't get DirId');
                                        LEAVE;
                                    END;
                                gCurrDirId := ioDrDirId;
                                ioVolIndex := 0;
                                gError := PBHGetVInfo(@fPBRec, FALSE);
                                IF (gError <> NoErr) THEN
                                    BEGIN
                                        ErrorOSErr('Couldn't get VInfo');
                                        LEAVE;
                                    END;
                                gCurrVRefNum := ioVRefNum;
                            END
                        ELSE
                            BEGIN
                                gCurrDirId := 2;
                                gCurrVRefNum := vRefNum;
                            END;
                        gCurrFilename := fName;
                        gCurrIndex := 0;
                        gCurrWRefNum := vRefNum;
                        IF (sPrevWRefNum <> gCurrWRefNum) THEN
                            BEGIN
                                CallPrevDirEnd;
                                BuildDirname;
                                DirectoryBegins;
                                sPrevDirId := gCurrDirId;
                                sPrevVRefNum := gCurrVRefNum;
                                sPrevWRefNum := gCurrWRefNum;
                            END;
                        ioDrDirId := gCurrDirId;
                        ioFDirIndex := gCurrIndex;
                        ioVRefNum := gCurrWRefNum;
                        gError := PBGetFInfo(@fPBRec, FALSE);
                        IF (gError <> NoErr) THEN
                            BEGIN
                                ErrorOSErr('Couldn't get FInfo');
                                LEAVE;
                            END;
                        gCurrFInfo := ioFInfo;
                        CallProcessFile;
                        IF gAbortPatrol THEN

```

```

LEAVE;
SFGetFile
(gSFGetPt, '', NIL, -1, gSFLst, NIL, gSFRep);
END;
gError := NoErr;
CallPrevDirEnd;
PatrolEnds;
END;
END;
END.

```

"SecurityPatrol.p"

```

PROGRAM SecurityPatrol(OUTPUT);
USES
  MemTypes, QuickDraw, OSIntf, ToolIntf, PackIntf, CodeSizeLimits, Globals,
  MainDlog, PasLibIntf, Patrol;
($R-)
CONST
  kPreprocessSelf = TRUE;
  kAwaitVerification = FALSE;
VAR
  gInitSecPatDone, gRptFileOpen: BOOLEAN;
  gScrDmpEnbPtr: Ptr;
  gScrDmpEnbSave: SignedByte;
  o: Text;
PROCEDURE PreprocessSelf; FORWARD;
PROCEDURE Writeln(pStr: Str255); FORWARD;
PROCEDURE Writeln(pChar: CHAR); FORWARD;
PROCEDURE Writeln; FORWARD;
PROCEDURE Writeln(pStr: Str255); FORWARD;
PROCEDURE Writeln(pNbr: LONGINT; pNbrDigits: INTEGER); FORWARD;
PROCEDURE Writeln(pType: ResType); FORWARD;
($Z*)
PROCEDURE ExitSecurityPatrol;
BEGIN
  IF gOption[eTrace] THEN
    Trace('ExitSecurityPatrol');
  gScrDmpEnbPtr := gScrDmpEnbSave;
  IF gRptFileOpen THEN
    Close(o);
  ExitToShell;
END;
($Z-)
PROCEDURE InitSecurityPatrol;
VAR
  sConfig: LONGINT;
BEGIN
  MaxApplZone;
  MoreMasters; MoreMasters; MoreMasters;
  gInitSecPatDone := FALSE;
  gRptFileOpen := FALSE;
  gScrDmpEnbPtr := Ptr(kScrDmpEnb);
  gScrDmpEnbSave := gScrDmpEnbPtr;
  gScrDmpEnbPtr := 0;
  Textbook(@thePort);
  Write ('SecurityPatrol is a Mac virus ');
  Write ('detector. ');
  TextFace([bold, extend]);
  Writeln('Use at your own risk. ');
  TextFace([]);
  Write ('The Save As... dialog below is ');
  Writeln('to save error reports. ');
  PLFflush(OUTPUT);
  PauseBriefly;
  InitGlobals;
  gOption[eBeeps] := TRUE; (override...)
  gDisabled[eFgPrC] := TRUE; (...defaults)
  InitMainDlog;
  InitPatrols;
  sConfig := BAnd(ORD4(Ptr(kSPConfig)^), $F);
  IF (sConfig = useFree)
  OR (sConfig = useAsync) THEN
    SFPutFile
      (gSFPutPt, 'Filename, or cancel to print',
       'SecurityPatrol Report', NIL, gSFRep)

```

```

ELSE
  SFPutFile
    (gSFPutPt, 'Filename, or cancel to quit',
     'SecurityPatrol Report', NIL, gSFRep);
WITH gSFRep DO
  IF good THEN
    BEGIN
      gCurrWDefNum := vRefNum;
      BuildDirname;
      Rewrite(o, CONCAT(gCurrDirname, fName));
    END
  ELSE
    IF (sConfig = useFree)
    OR (sConfig = useAsync) THEN
      Rewrite(o, 'PRINTER: ')
    ELSE
      BEGIN
        Writeln('Run cancelled. ');
        PLFflush(OUTPUT);
        PauseBriefly;
        ExitSecurityPatrol;
      END;
  gRptFileOpen := TRUE;
  Writeln ('This copy of Security Patrol 1.0 ');
  Writeln ('is being maintained by ');
  TextFace([bold, extend]);
  gPgmname := '<<your name here>>';
  Writeln(gPgmname);
  TextFace([]);
  Writeln ('The following run was done on ');
  GetTime(gDateTimeRec);
  WITH gDateTimeRec DO
    BEGIN
      year := year mod 100;
      Writeln (month, 2);
      WritelnChar('/');
      Writeln (day, 2);
      WritelnChar('/');
      Writeln (year, 2);
      Writeln (' at ');
      Writeln (hour, 2);
      WritelnChar(':');
      IF minute < 10 THEN
        WritelnChar('0');
      Writeln (minute, 1);
      WritelnChar(':');
    END;
  IF kPreprocessSelf THEN
    PreprocessSelf
  ELSE
    Writeln('Didn't perform self-tests. ');
  gInitSecPatDone := TRUE;
END;
PROCEDURE PreprocessSelf;
VAR
  i: INTEGER;
  sCIPtr, sEntActual, sEntShouldB, sOffActual, sOffShouldB: LONGINT;
  sResType: ResType;
  sSave: TMainOpt;
PROCEDURE Abort(pStr: Str255);
BEGIN
  ErrorBegins(pStr);
  Writeln;
  CommentBegins;
  Writeln('Assuming infected. ');
  CommentBegins;
  Writeln ('Contact ');
  Writeln (gPgmname);
  Writeln (' to be sure. ');
  CommentBegins;
  Writeln ('(These msgs apply to ');
  Writeln (gCurrFilename);
  Writeln (' itself, not to your system. ');
  CommentBegins;

```

```

gOption[eAwait] := TRUE;
gOption[eBeeps] := TRUE;
ErrorEnds(4);
ExitSecurityPatrol;
END;
BEGIN
BlockMove(@gOption,@sSave,SIZEOF(TMainOpt));
ZeroOut (@gOption, SIZEOF(TMainOpt));
gOption[eRmVir] := TRUE;
( gOption[eTrace] := TRUE; )
IF gOption[eTrace] THEN
Trace('PreprocessSelf');
GetCodeSizeLimits;
GetRsrc(@gCode0,'CODE',0,ResId);
IF (gCode0.fFlag <> kRsrcHdlValid) THEN
Abort('Unable to get own CODE 0');
IF NOT(Code0IsValid) THEN
Abort('Found unexpected CODE 0 header');
LookForKnownViruses;
FOR i := 1 TO Count1Types DO
BEGIN
Get1IndType(sResType,i);
IF (sResType = 'SIZE') THEN
BEGIN
IF (Count1Resources('SIZE') > 1) THEN
Abort('Too many SIZE resources');
GetRsrc(@gCurrRsrc,'SIZE',1,Index);
IF (gCurrRsrc.fSize > 10) THEN
Abort('SIZE resource too large');
ReleaseRsrc(@gCurrRsrc);
END
ELSE IF (sResType <> 'CODE') THEN
BEGIN
ErrorBegins('Found a rsrc of type ');
Writeln(sResType);
ErrorEnds(0);
Abort ('Only CODE and SIZE allowed');
END;
END;
WITH TJTHdl(gCode0.fHdl)^,fJTEntry[1] DO
BEGIN
IF (fNbrBytesInTable <> gJTSIZE) THEN
BEGIN
ErrorBegins('Jump table size is ');
Writeln(fNbrBytesInTable,1);
Writeln (' , should be ');
Writeln (gJTSIZE,1);
ErrorEnds(0);
Abort ('Invalid Jump Table size');
END;
IF NOT(JTEIsValid(@fJTEntry[1])) THEN
Abort('Invalid Jump Table entry');
IF (fSegId <> 1) THEN
BEGIN
ErrorBegins('Enters at CODE ');
Writeln(fSegId,1);
Writeln (' , should enter at CODE 1');
ErrorEnds(0);
Abort ('Invalid start address');
END;
sOffActual := 4 + fOffset;
END;
WITH gCurrRsrc DO
BEGIN
GetRsrc(@gCurrRsrc,'CODE',1,ResId);
IF (fFlag <> kRsrcHdlValid) THEN
Abort('Couldn't look at own CODE 1');
sC1Ptr := BAND($00FFFFFF,ORD4(fHdl^));
sEntActual := sC1Ptr + sOffActual;
IF (TWordPtr(sEntActual)^ = $4EFA) THEN
BEGIN
sOffActual :=
sOffActual+2+TWordPtr(sEntActual+2)^;
sEntActual := sC1Ptr + sOffActual;

```

```

END;
sEntShouldB := gEntryPoint;
sOffShouldB := sEntShouldB - sC1Ptr;
IF (sEntActual <> sEntShouldB) THEN
BEGIN
ErrorBegins('Invalid start address');
Writeln;
CommentBegins;
Writeln ('Enters at address $');
ShortHexDump(@sEntActual,4);
Writeln (' (CODE 1 + $)');
ShortHexDump(@sOffActual,4);
Writeln (' ) -> $');
ShortHexDump(Ptr(sEntActual),4);
Writeln;
CommentBegins;
Writeln ('Should enter at $');
ShortHexDump(@sEntShouldB,4);
Writeln (' (CODE 1 + $)');
ShortHexDump(@sOffShouldB,4);
Writeln (' ) -> $');
ShortHexDump(Ptr(sEntShouldB),4);
Writeln;
CommentBegins;
Writeln ('CODE 1 begins at $');
ShortHexDump(@sC1Ptr,4);
ErrorEnds(0);
Abort ('This is not a user error');
END;
ReleaseRsrc(@gCurrRsrc);
END;
ReleaseRsrc(@gCode0);
IF (Count1Resources('CODE') > gMaxCode+1) THEN
Abort('Too many CODE resources. ');
IF kAwaitVerification THEN
BEGIN
ErrorBegins('The following are the ');
Writeln ('fingerprints' of ');
Writeln (gCurrFilename);
Writeln (' itself: ');
ErrorEnds(0);
END;
FOR i := 0 TO gMaxCode DO
WITH gCurrRsrc DO
BEGIN
GetRsrc(@gCurrRsrc,'CODE',i,ResId);
IF (fFlag <> kRsrcHdlValid) THEN
Abort('Couldn't look at next CODE');
IF (fSize > gSizeLimit[i]) THEN
BEGIN
ErrorBegins('Failed a CODE size test');
Writeln;
CommentRsrcBegins(@gCurrRsrc);
Writeln (' size is ');
Writeln(fSize,1);
Writeln (' , which exceeds its ');
Writeln(gSizeLimit[i],1);
Writeln (' size limit. ');
Abort('May contain an imbedded virus');
END;
IF kAwaitVerification THEN
BEGIN
ProcessCurrRsrc;
CommentFgPrRsrc(@gCurrRsrc);
END;
ReleaseRsrc(@gCurrRsrc);
END;
IF kAwaitVerification THEN
BEGIN
ErrorBegins('Time to make a decision: ');
Writeln;
CommentBegins;
Writeln('Verify fingerprints if you can');
CommentBegins;

```

```

Writeln('Press command-period to abort');
CommentBegins;
Writeln('Press any other key to continue');
CommentBegins;
gOption[eAwait] := TRUE;
ErrorEnds(0);
IF gAbortPatrol THEN
BEGIN
PauseBriefly;
ExitSecurityPatrol;
END;
END;
Writeln('Passed all current self-tests. ');
PauseBriefly;
BlockMove(@sSave, @gOption, SIZEOF(TMainOpt));
END;
($Z*)
PROCEDURE WriteFilenameToReport;
BEGIN
WITH gReportFlags DO
IF NOT(fWroteFilename) THEN
BEGIN
IF NOT(fWroteDirname) THEN
BEGIN
Writeln(o, gCurrDirname);
fWroteDirname := TRUE;
END;
Write(o, gInd, gCurrFilename);
IF gActiveSelf THEN
BEGIN
WriteCo, ' (Active Self)';
IF gInitSecPatDone
AND NOT(kProcessSelf) THEN
WriteCo, ' skipped';
END
ELSE IF gActiveSys THEN
WriteCo, ' (Active System)';
Writeln(o);
fWroteFilename := TRUE;
END;
END;
PROCEDURE WriteFilenameToScreen;
BEGIN
WITH gScreenFlags DO
IF NOT(fWroteFilename) THEN
BEGIN
IF NOT(fWroteDirname) THEN
BEGIN
Writeln(gCurrDirname);
fWroteDirname := TRUE;
END;
Write(gInd, gCurrFilename);
IF gActiveSelf THEN
BEGIN
Write(' (Active Self)');
IF gInitSecPatDone
AND NOT(kProcessSelf) THEN
Write(' skipped');
END
ELSE IF gActiveSys THEN
Write(' (Active System)');
Writeln;
PLFflush(OUTPUT);
fWroteFilename := TRUE;
END;
END;
PROCEDURE Writeln(pStr: Str255);
BEGIN
Writeln(pStr);
IF gRptFileOpen THEN
WriteCo, pStr);
END;
PROCEDURE WritelnChar(pChar: CHAR);
BEGIN

```

```

Write(pChar);
IF gRptFileOpen THEN
WriteCo, pChar);
END;
PROCEDURE Writeln;
BEGIN
Writeln;
PLFflush(OUTPUT);
IF gRptFileOpen THEN
BEGIN
WritelnCo;
END;
END;
PROCEDURE WritelnFilename;
BEGIN
IF gRptFileOpen THEN
WriteFilenameToReport;
WriteFilenameToScreen;
END;
PROCEDURE WritelnFilenameToScreenOnlyForNow;
BEGIN
WriteFilenameToScreen;
END;
PROCEDURE Writeln(pStr: Str255);
BEGIN
Writeln(pStr);
PLFflush(OUTPUT);
IF gRptFileOpen THEN
BEGIN
WritelnCo, pStr);
END;
END;
PROCEDURE WritelnNbr(pNbr: LONGINT; pNbrDigits: INTEGER);
BEGIN
Write(pNbr: pNbrDigits);
IF gRptFileOpen THEN
WriteCo, pNbr: pNbrDigits);
END;
PROCEDURE WritelnType(pType: ResType);
BEGIN
Writeln(pType);
IF gRptFileOpen THEN
WriteCo, pType);
END;
PROCEDURE zzSecurityPatrol;
BEGIN
END;
BEGIN
InitSecurityPatrol;
WHILE TRUE DO
BEGIN
gAbortPatrol := FALSE;
CASE MainDlogWorkRequested OF
eDirs: PatrolDirectories (FALSE);
eDiry: PatrolDirectories (TRUE);
eEvery: PatrolEverything;
eFiles: PatrolFiles;
OTHERWISE LEAVE;
END; (CASE)
END;
Writeln;
Writeln('*****');
Writeln;
Writeln('Totals over all patrols: ');
ListCounts(@gTotals);
ExitSecurityPatrol;
END.
SecurityPatrol.Link (TML 2.5 only)
!PAS$Xfer
SecurityPatrol
PAS$Library
MacIntf
MacIntfGlue

```

```

BitProcs
CodeSizeLimits
Globals
MainDlog
PasLibIntf
Patrol
<
Globals/Globals
<
BitProcs/Fingerprint
Globals/Fingerprint
/End

SecurityPatrol.Proj (TML II Only)
SecurityPatrol.p.o f d
  CodeSizeLimits.p d
  Fingerprint.ipas d
  Globals.p d
  MainDlog.p d
  Patrol.p d
  SecurityPatrol.p
    TMLPascal SecurityPatrol.p
CodeSizeLimits.p.o f d
  CodesSizeLimits.p
    TMLPascal CodeSizeLimits.p
Globals.p.o f d
  Fingerprint.ipas d
  Globals.p
    TMLPascal Globals.p
MainDlog.p.o f d

```

```

Globals.p d
MainDlog.p
  TMLPascal MainDlog.p
Patrol.p.o f d
  Globals.p d
  Patrol.p
    TMLPascal Patrol.p

SecurityPatrol ff d
  CodeSizeLimits.p.o d
  Globals.p.o d
  MainDlog.p.o d
  Patrol.p.o d
  SecurityPatrol.p.o
    Link -w -t 'APPL' -c '????' d
      -ra Fingerprint=$14 d
      -ra Globals=$14 d
      -ra Main=$14 d
    SecurityPatrol.p.o d
    CodeSizeLimits.p.o d
    Globals.p.o d
    MainDlog.p.o d
    Patrol.p.o d
    "(Libraries)"Runtime.o d
    "(Libraries)"Interface.o d
    "(Libraries)"ObjLib.o d
    "(TMLPLibraries)"TMLPasLib.o d
    "(TMLPLibraries)"SANELib.o d
    -o SecurityPatrol

```



Pascal Procedures

How To Write a Font Tool for MPW



Randy Leonard
Jacksonville, FL
Volume 5, Number 3

[Randy Leonard is currently employed by TML Systems, Inc. while completing his Master's thesis in computer science at the University of Central Florida. His thesis involves improving the efficiency of several key algorithms in constructive solid geometry (CSG). The thesis is written entirely with TML Pascal II and he uses several of his own MPW Tools to aid in his development.]

The introduction of the Macintosh Programmer's Workshop (MPW) v3.0 this January requires that we revisit this programming environment and study its new and exciting features as well as the significant features of previous versions of the product. This article discusses how the MPW environment can be enhanced with the addition of your own integrated programming tools. The type of commands introduced here are capable of executing in the background while running MPW v3.0. This capability enables you to continue your most important task in the foreground while such tasks as compiling and linking of programs occur in the background. Indeed, the tool introduced here may also run in the background.

Macintosh Programmer's Workshop

The Macintosh Programmer's Workshop is the official Macintosh development environment from Apple Computer, Inc. The MPW Shell is a complete development system for the Macintosh that includes, among other things, a multi-windowing text editor and a command processor. Also included with MPW is a linker, make facility, resource compiler, resource decompiler, source code management system, and more. There is also complete online help as well as an optional graphical interface for just about every command available in MPW.

MPW has often times been accused of having a steep learning curve. The author believes this false accusation can be based upon two pretexts. The first is due to unfamiliarity with the product *[if you come from Big Blue or have been in VAXinated, it might be familiar, but still not easy-ed]*. The second is due to the extraordinary power that MPW can provide its advanced users via the command line environment and MPW's advanced command processor. The truth of the matter is that MPW is as easy to learn and use as any other programming environment if you were to utilize only those features found in MPW that the other programming environments provide.

The true beauty of MPW is found in its open architecture. That is, anyone can expand the functionality of MPW to suit his own needs. There exist two methods of adding new commands to MPW: scripts and tools. This two part article will demonstrate how to create new programming tools and how to fully integrate these tools into the MPW environment. This month's article will give the motivation for writing your own MPW tool and show

how it is done. Part two, to appear next month, will demonstrate how to integrate the tool into the MPW's online help facility and how to develop a graphical interface for the new command.

MPW Tools

Tools provide a method of adding commands that are not inherent to the MPW command processor. For example, the TML Pascal II compiler is an extension to the MPW environment and is therefore implemented as a tool. Other examples of MPW Tools are the canonical speller Canon and TMLPasMat. TML Systems, in fact, used these two tools to put the finishing touches on both its Source Code Library II and example programs found on the TML Pascal II distribution disk. Canon adjusts all identifiers in source code files so they have the same capitalization as found in *Inside Macintosh*. TMLPasMat will format all Pascal source code files to a consistent format that you define. These two tools were especially helpful since programming styles of the programmers at TML Systems vary.

However, we needed even more help in providing a consistent format for all source code files that left our office. As it turns out, some programmers at TML Systems prefer different fonts and different font sizes for their work. Some programmers would print their work on the laser writer reduced to 75% while others would print at 100%. Some prefer different tab settings and still others have large screens on which to edit their programs. This last problem could be especially annoying to our customers with small screens since only a small part of the window of a text file would appear on their screen when they opened the file. Each customer could easily resolve the problem by resizing the window for his screen, but what if the window appeared off-screen to begin with!

Another problem unrelated to each programmer's taste was the problem of USYM resources. The TML Pascal II compiler writes symbol table information for separately compiled units to the resource fork of the main unit source code file as USYM resources. This is perhaps a more favorable solution than creating separate files to hold the information as other compilers do. Deleting a file that contains symbol table information is easy, but how do you easily erase resources from the resource fork of a source code file?

To solve the above problems, we developed an MPW tool that changed the font of MPW text files to any specified font and font size. Tab spacing is set to any defined value and the window is made to appear on any Macintosh screen with the top left corner slightly staggered from other windows and the bottom right corner extending to the bottom right of the screen. The page setup is reset to print at 100%. All USYM and other resources are optionally deleted as well. We call this tool ChangeTextRes.

Below, we explain how to implement this tool, but first a little background.

What Every Tool has In Common

MPW Tools are usually invoked from the command line of the MPW Shell. The command line of an MPW command is simply the line on which the MPW command exists. Parameters are passed to the tool by placing them on the command line. A graphical interface for the tool, called the Commando, does exist, but is not typically used. The Commando interface for a tool is invoked by typing in the command name followed by the ellipses character (...). This character is obtained by holding the option key and pressing the semicolon. The implementation of a custom Commando interface for ChangeTextRes is discussed in next month's article.

Apple has defined several conventions for MPW Tools that allow them to work well together in an integrated fashion. First and most important is that a tool have some default behavior. Deviations from this default should result only from options specified on the command line. A command line option is simply a parameter found on the command line starting with the character '-'. For example, the resource decompiler tool DeRez is invoked by typing:

```
DeRez filename
```

where filename specifies any resource file at all. If you wish only to decompile only dialog resources, you would type:

```
DeRez filename -only DLOG
```

Now, only resources of type 'DLOG' will be decompiled. The behavior of the DeRez tool has been modified by the -only option. Many command line options may be specified, but the order in which they are given should never matter.

Another rule is that tools are to run silently, they should require no interaction with the user to carry out its task. The only visual feedback from the tool should be in the form of a spinning cursor. The reasoning for these rules have their roots in the advanced capabilities of the MPW command processor. Should the reader wish to become a more powerful user of MPW, he is referred to the chapter "Using the Command Language" of the *Macintosh Programmer's Workshop Reference*.

The Command Processor

Each time a command is entered, the MPW command processor attempts to interpret it. If it is unsuccessful, the command processor assumes the command is either a tool, script, or an external application. Since we are writing a tool, we do not need to concern ourselves to much with the command processor, but there are a few features that need to be discussed.

Before invoking an MPW tool, the command processor first attempts to interpret parts of the command line. Certain characters have special meaning to the command processor. For example, the ~ (Option-x) character is a wild card character. If ~.p is specified, the command processor will expand this to a

sequence of filenames that end with .p. The MPW Tool never sees the ~.p parameter. Rather, it sees the sequence of filenames that match the ~.p pattern! This not only increases a user's productivity, but also simplifies the writing of tools.

Also allowed are special characters for I/O redirection, piping, and substitution of MPW Shell variables[*very nice to have.-ed*]. Knowledge of these features are not necessary for the average MPW user. See the *Macintosh Programmer's Workshop Reference* for more details.

Three file variables are predefined for MPW Tools, *input* (standard input), *output* (standard output), and *diagnostic*. As expected, standard input is the Macintosh keyboard. Standard output and diagnostic output are both the current topmost window found in the MPW environment.

How ChangeTextRes Works

The underlying theory of ChangeTextRes is quite simple. The tool receives from the command line the name of each file it is to work on as well as any command line options that may exist. The resource fork of each MPW text file specified on the command line is deleted if the command option -d is also present on the command line. If the -d option is not specified then no resources of any files are ever deleted. ChangeTextRes will then change the file's font, font size, and tab setting.

If ChangeTextRes is instructed to delete the entire resource fork with the -d option, there will no longer be any resource specifying the page setup or window size and position. If a file has no resource for page setup or file window position, the MPW Shell will assume default values. The default value for window position is to stagger the window's top left corner and cover the rest of the screen. The -d option will delete all USYM resources created by the TML Pascal II compiler as well.

ChangeTextRes assumes default values of Monaco, 9 point, and 3 for the font, font size, and tab setting, respectively. The user may change these values with the command line options -f, -s, and -t. The -f option, followed by a font name instructs ChangeTextRes to use that font name. The -s option followed by an integer value instructs ChangeTextRes to use the specified font size. And the -t option followed by an integer value instructs ChangeTextRes to use the specified tab setting. An error message is generated if an invalid font, a font size less than 1 or greater than 127, or a tab setting less than 1 or greater than 24 is specified.

To use ChangeTextRes, type the command followed by all file names and options desired. Keep in mind the MPW Shell's filename expansion capabilities discussed above. For example:

```
ChangeTextRes MyProject.proj ~.p  
-f Courier -s 12 -t 4 -d
```

will delete all resources of all MPW text files ending with a .p as well as the file MyProject.proj. The font for each file is set to Courier 9 point and the tab setting is set to 4.

Accessing the Command Line

Accessing the command line parameters from within a tool

is quite simple. The IntEnv unit defines two global variables used to access the command line parameters: argv and argc. The argv variable is a pointer to an array of pointers to strings. Argc tells how many parameters exist in the argv array. The argv array starts at element 0 and argc is always one greater than the actual number of parameters found on the command line. The expression argv[argc] is always equal to nil. The first parameter of the argv array (argv[0]) is the name of the MPW tool itself and is useful for error message generation. All parameters to the tool are stored in argv[1]^ to argv[argc-1]^ . It is up to the programmer to read in command line parameters and to distinguish between regular parameters and command line options. The ChangeTextRes tool contains the procedure ReadCommandLine to accomplish this task.

```

procedure ReadCommandLine;
var
  argvIndex   : integer;
  arg         : Str255;
begin
  if argc = 1 then SyntaxError(9, '');
  argvIndex := 1;
  while argvIndex < argc do begin
    arg := argv[argvIndex];
    if length(arg) <> 0 then
      if arg[1] = '-' then
        if length(arg) > 1 then
          HandleOption(arg, argvIndex)
        else SyntaxError(8, '');
      argvIndex := argvIndex + 1;
    end; { while }
  end;
end;

```

In this routine, argvIndex is used to traverse the command line parameters. Each time a command line option is found, the procedure HandleOption is called. HandleOption will read the option and set program global variables accordingly. Since options may require an additional parameter (e.g. the ChangeTextRes option -f, -s, and -t), HandleOption must have the ability to read the next parameter(s) on the command line and increment argvIndex accordingly. If an invalid command line option or invalid value corresponding to a valid option is found, HandleOption will generate an error and terminate the program.

Rewriting a File's Resource Fork

The sole purpose for ChangeTextRes is to rewrite part or all of a file's resource fork. Since it is intended to work only on source code files, the segment of the program that effects a file's resource fork must first check to see if the file is of type 'TEXT' and has a creator of 'MPS ' (the MPW signature). If this is so, ChangeTextRes will proceed to change the file's resource fork. Below is the segment of code that accomplishes this task.

```

if (finderInfo.fdType = 'TEXT') and
   (finderInfo.fdCreator = 'MPS ') then begin
  if gResDelete then begin
    anOSError := OpenRF(filename, vRefNum,
                        ResRefNum);
    anOSError := SetEOF(ResRefNum, 0);
    anOSError := FSClose(ResRefNum);
  end;
  result := IEFAccess(filename, F_STabInfo,

```

```

                                gTabSize);
    result := IEFAccess(filename, F_SFontInfo,
                                arg);
end

```

The global variable gResDelete is affected by the -d option. If the -d option is present on the command line, gResDelete is set to true, otherwise it is set to false. Only when gResDelete is true will all the resources be deleted.

An MPW file's font, font size and tab setting are modified with calls to the IEFAccess function. This function is defined in the IntEnv unit. There are three parameters to IEFAccess. The first is the filename on which the routine is to operate. The second parameter defines which operation to perform, and the third provides a means of passing data to and receiving results from the IEFAccess routine.

Our first call to IEFAccess sets the tab setting of a file. The predefined constant F_STabInfo informs IEFAccess to set the tab of the specified file and the third parameter specifies the desired tab setting. The second call to IEFAccess sets the font and font size. The second parameter of IEFAccess is set to the predefined constant F_SFontInfo. Apple has inadvertently documented that the third parameter, in this case, is a pointer to the new font and font size. This is not the case, rather, the upper word of this long integer is the font number and the lower word contains the font size. Both F_STabInfo and F_SFontInfo are defined in the IntEnv unit.

Spinning the Beach Ball Cursor

MPW Tools are, by convention, supposed to provide visual feedback to the user by displaying and spinning a cursor. By default, this cursor is the MPW beach ball cursor but may be any other cursor the programmer defines. Spinning cursors have a resource type of 'acur' and may be created with MPW's resource editor and/or resource compiler. See the appendix "Programming for the Shell Environment" in the *MPW Pascal Reference Manual* or Appendix D of the *TML Pascal II Language Reference* for more details on creating and using such resources.

All the routines related to the operation of the cursor by an MPW Tool are contained in the CursorCtl unit. Only two routines in this file are required by most tools: InitCursorCtl and RotateCursor.

For a tool to rotate the cursor, it must first call InitCursorCtl. This routine has just one parameter which is a handle to an 'acur' resource. If this parameter is nil, then the cursor defaults to MPW's spinning beach ball cursor. Call this routine very early in the program to prevent fragmentation of the heap.

Initializing the spinning cursor does not in itself cause the cursor to spin. The MPW Tool must manually spin the cursor itself. This may seem to be an inconvenience, but it is actually to your advantage. For example, a user can track progress of the MPW Linker by watching which way the beach ball rotates. The linker has three phases, each change in phase is accompanied by a change in direction of rotation of the cursor.

To rotate the cursor, call RotateCursor(value) where value is some integer or long integer. Each time this procedure is called, value is added to an internal counter. When this counter is an

even increment of 32, the beach ball is rotated. If *value* is positive, the cursor is rotated in the clockwise direction. If *value* is negative, the cursor is rotated in the counter-clockwise direction. It is important to call RotateCursor on a frequent basis. It is usually best to place this procedure call in the main loop of the program.

Software Interrupts

MPW Tools should be capable of responding to software interrupts known as signals. Currently, only one type of signal exists and that is the command-period (.). Signals have the capability of pre-empting a tool or any other MPW command. Tools will automatically respond to a signal but it may be necessary at times to prevent a signal from pre-empting a tool. Several routines in the Signal unit allow a tool to control the effect a signal may have on it. ChangeTextRes does not in any way attempt to control a signal's effect.

The default actions taken by a tool in response to a signal are to close all open files, execute any installed exit procedures, and terminate the program. See the appendix "Programming for the Shell Environment" in the *MPW Pascal Reference Manual* or Appendix D of the *TML Pascal II Language Reference* for more details on how to install exit procedures. Also refer to these manuals for information on how to prevent or delay the effects of signals on MPW Tools.

Returning Status Results

There are basically three different conditions that cause ChangeTextRes to terminate. The first is normal termination and arises when ChangeTextRes has successfully completed processing its data. There are two abnormal termination conditions. One is due to invalid syntax of command line parameters and the other to the inability for the tool to successfully complete its task. In any case, when a tool returns control to the MPW Shell, it must inform the Shell of its termination condition. This is done by returning a status code.

Defined in the IntEnv unit is the procedure IEEExit. This procedure has one parameter of type LongInt. When a tool is to terminate, either normally or abnormally, it should call IEEExit. Note that IEEExit actually terminates the program. The value of its parameter is returned to the MPW Shell as a status code, which by convention, is zero to signify normal completion and non-zero to indicate abnormal termination. ChangeTextRes returns 1 to indicate a syntax error and 2 to indicate other errors.

Further Reading

This article has addressed many of the issues involved in writing MPW Tools, but there still remains a significant amount of potential yet to be realized. The chapter "Writing an MPW Tool" of the *Macintosh Programmer's Workshop Reference* discusses all the issues of writing MPW Tools, and does so in much greater detail than presented here. The chapter "Building an Application, a Desk Accessory, or an MPW Tool" of the same reference describes the mechanics of building a tool, but this knowledge is not necessary if you are using the TML Project Manager.

Chapter 8 of the *TML Pascal II User's Guide* shows how to write MPW Tools as does *Programming with Macintosh Programmer's Workshop*, by Joel West. This second book is highly recommended for any MPW user. Appendix D of the *TML Pascal II Language Reference* and appendix titled "Programming for the Shell Environment" of the *MPW Pascal Reference Manual* give complete descriptions of the interface files required to create MPW Tools.

Next Month

Next month, we will develop a graphical interface, called the Commando interface, for the ChangeTextRes tool. We will also show how to add or modify Commando interfaces to other existing MPW Tools.

```

program ChangeTextRes;
( ChangeTextRes.p

    An MPW Tool to delete resource fork of MPW
    text files and rewrite the resource fork
    to specify a desired tab setting, font,
    and font size.
    (c) TML Systems, Inc., 1988
    All rights reserved. Publication rights granted to
    MacTutor.
)

uses MemTypes, QuickDraw, OSIntf, ToolIntf,
    PackIntf, PasLibIntf,

    ( required for MPW Tools )
    CursorCtl, IntEnv;

var
    ResRefNum : integer;
        ( reference number for resource fork of a given file )
    filename : Str255;
    aStringPtr : StringPtr;
        ( reference number for default drive )
    vRefNum : integer;
        ( Finder information for a given file )
    fnderInfo : FInfo;
        ( result from Mac ROM file I/O calls )
    anOSErr : OSErr;
        ( passed to IEFAccess specifies font and font size )
    arg : LongInt;
        ( result from IEFAccess calls )
    result : LongInt;
    i : integer;

    ( Font number of specified font as returned by GetFNum )
    gFont : integer;
    gFontSize : LongInt;
    gTabSize : LongInt; ( tab setting )
        ( delete all of file's resources? )
    gResDelete : boolean;

function UpperCase(str: Str255): Str255;
( Convert an alphanumeric string to all
  uppercase characters.
)
var
    i: integer;
begin
    for i := 1 to length(str) do
        if (str[i] >= 'a') and
            (str[i] <= 'z') then
            str[i] := chr(ord(str[i]) - 32);
        UpperCase := str;
    end;
end;

```

```

procedure SyntaxError(err: integer;
                      msg: Str255);
( Display the appropriate syntax error and
  then exit from the program. Return a
  status value of 1 indicating an early
  termination of program.
)
begin
  case err of
    1: writeln('# ', msg, ' is an invalid option');
    2: writeln('# missing font');
    3: writeln('# missing font size');
    4: writeln('# missing tab setting');
    5: writeln('# ', msg, ' is an invalid font');
    6: writeln('# ', msg, ' is an invalid font size');
    7: writeln('# ', msg, ' is an invalid tab size');
    8: writeln('# the - character must be
        accompanied by an option');
    9: begin
        writeln('# Usage - ChangeTextRes [name...] ');
        writeln('      -f fontname      # set
        font of files to fontname');
        writeln('      -s fontsize     # set
        font size of files to fontsize');
        writeln('      -t tabs          # set
        tab setting to tabs');
      end;
    otherwise
      writeln('fatal error #', err);
    end;
    IEEExit(1); ( return error status of 1 )
  end;

procedure HandleOption(opt: Str255;
                      var argIndex: integer);
(
  Set the appropriate global flag for each
  command line option encountered on the
  command line. If an invalid
  option is found, give an error message and
  exit from the program. If the option
  requires an additional command line
  parameter (e.g. -f Monaco), then retrieve
  the option(s) needed and increment the
  argIndex counter appropriately.
)
var
  NumString, str      : Str255;
begin
  str := UpperCase(opt);
  Delete(str, 1, 1);
    (delete the '-' character)
  if str = 'F' then begin ( set font )
    argIndex := argIndex + 1;
    if argIndex < argc then begin
      GetFNum(argv[argIndex], gFont);
      if gFont < 0 then
        SyntaxError(5, argv[argIndex]);
      end
    else SyntaxError(2, '');
  end
  else if str = 'S' then begin
    ( set font size )
    argIndex := argIndex + 1;
    if argIndex < argc then begin
      StringToNum(argv[argIndex], gFontSize);
      if (gFontSize <= 0) or
        (gFontSize >= 128) then begin
        NumToString(gFontSize, NumString);
        SyntaxError(6, NumString);
      end;
    end
  else SyntaxError(3, '');

```

```

end
else if str = 'T' then begin ( set tab )
  argIndex := argIndex + 1;
  if argIndex < argc then begin
    StringToNum(argv[argIndex], gTabSize);
    if (gTabSize <= 0) or
      (gTabSize >= 25) then begin
      NumToString(gFontSize, NumString);
      SyntaxError(7, NumString);
    end;
  end
  else SyntaxError(4, '');
end
else if str = 'D' then
  gResDelete := true
else SyntaxError(1, str);
end;

procedure SkipOption(opt: Str255;
                    var argIndex: integer);
(
  This routine is called only after the
  command line parameters have already been
  scanned once using HandleOption. The
  purpose of this routine is to properly
  increment argIndex according to the
  appropriate command line options.
)
var
  str: Str255;
begin
  str := UpperCase(opt);
  Delete(str, 1, 1);
    (delete the '-' character)
  if str = 'F' then ( set font )
    argIndex := argIndex + 1
  else if str = 'S' then ( set font size )
    argIndex := argIndex + 1
  else if str = 'T' then ( set tab size )
    argIndex := argIndex + 1
  else if str = 'D' then
    ( nothing )
  end;

procedure ReadCommandLine;
var
  argVIndex      : integer;
  arg             : Str255;
begin
  if argc = 1 then SyntaxError(9, '');
  argVIndex := 1;
  while argVIndex < argc do begin
    arg := argv[argVIndex];
    if length(arg) <> 0 then
      if arg[1] = '-' then
        if length(arg) > 1 then
          HandleOption(arg, argVIndex)
        else SyntaxError(8, '');
    argVIndex := argVIndex + 1;
  end; ( while )
end;

procedure ReportError(error: integer;
                    filename: Str255);
(
  Generate the appropriate error message
  then exit from the program. Return a
  status value indicating early termination
  from the program.
)
begin
  if error = 0 then
    exit(ReportError);
  write(diagnostic, 'ERROR! ');

```

```

case error of
-35: writeln(diagnostic, filename,
            ' volume does not exist');
-36: writeln(diagnostic, filename,
            ' IO Error');
-37: writeln(diagnostic, filename,
            ' is a bad filename or volume name');
-42: writeln(diagnostic,
            'Too many files open');
-43: writeln(diagnostic, filename,
            ' not found');
-45: writeln(diagnostic, filename,
            ' is locked');
-46: writeln(diagnostic, filename,
            ' is locked by a software flag');
-47: writeln(diagnostic, filename,
            ' is busy; one or more files are open');
-53: writeln(diagnostic, filename,
            ' volume not on-line');
-54: writeln(diagnostic, filename,
            ' cannot be opened for writing, file is
              locked');
-61: writeln(diagnostic, filename,
            ' Read/write permission doesn't allow writing');
otherwise
    writeln(diagnostic, 'OS error #',
            error, ' has occurred. ');
    writeln(diagnostic, ' Reference Inside
            Macintosh pp. III:205-209 for further details');
end;
IEExit(2);
end;

begin (main program)
    ( make first stmt to avoid heap fragmentation )
    InitCursorCtl(nil);
    InitFonts; (so we can read in font names)

    ( so we read in JUST the font names! )
    SetResLoad(false);

    ( Set default values )
    gResDelete := false;
    gFont := 4;
    gFontSize := 9;
    gTabSize := 3;

```

```

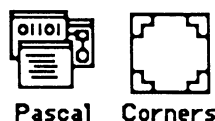
ReadCommandLine;
arg := gFont;
arg := BSL(arg, 16);
arg := arg + gFontSize;
anOSError := GetVol(aStringPtr, vRefNum);
if anOSError <> 0 then
    ReportError(anOSError, aStringPtr);
i := 1;
while i < argc do begin
    ( Make cursor rotate each time through loop )
    RotateCursor(32);
    filename := argv[i];
    if length(filename) = 0 then begin
        i := i + 1;
        cycle;
    end;
    if filename[1] = '-' then
        SkipOption(filename, i)
    else begin ( valid filename )
        anOSError := GetFInfo(filename, vRefNum, fnderInfo);
        if anOSError <> 0 then begin
            ReportError(anOSError, filename);
            cycle;
        end
        else begin ( file exists )
            if (fnderInfo.fdType = 'TEXT') and
                (fnderInfo.fdCreator = 'MPS ')
                then begin
                if gResDelete then begin
                    anOSError := OpenRF(filename, vRefNum, ResRefNum);
                    anOSError := SetEOF(ResRefNum, 0);
                    anOSError := FSClose(ResRefNum);
                end;
                result := IEFAccess(filename, F_STabInfo, gTabSize);
                result := IEFAccess(filename, F_SFontInfo, arg);
            end
            else
                writeln('WARNING! ', filename, ' is not an MPW text
                file, resources not deleted');
            end; ( file exists )
            end; ( valid filename )
            i := i + 1;
        end; ( while i < argc )
        writeln;
        SetResLoad(true);
        IEExit(0); ( Normal status return )
    end. (main program)

```



Advanced Mac'ing

Re-sizing with Window Corners



Clifford Story
Goleta, CA
Volume 5, Number 4

I suppose you've probably heard of the new Open Look interface for Unix. (Not so new anymore, actually!) The thing looks pretty familiar — windows, scroll bars, menus. Everything has some kind of a twist on it, though. The menus are all pop-ups; the scroll bars include the up and down arrows as part of the indicator; and the windows can be re-sized from any corner, not just the lower right corner.

That's an interesting idea, re-sizing from any corner. I wonder if you could do that on the Mac... I held a long argument with myself on that topic, and the affirmative finally won. You can do it on the Mac; here's one way.

Before I get started, let me thank Tom Leonard, who encouraged me, and Don Melton and Mike Ritter, whose article in the April MacTutor inspired me.

THE WINDOW

The first half of the problem is to draw the window with appropriate grow brackets in the corners, and to have it return the "wInGrow" message when the user clicks in one of them. This is done in a window definition procedure, or WDEF.

WDEF's have been pretty well covered in MacTutor; I refer you to the Window Manager chapter of Inside Macintosh, and to Melton and Ritter's WDEF discussed in their April article and published in May.

By the way, an earlier version used WDEF 0 to do most of the work. Apple, however, has declared its intention to remove WDEF 0 from the System file (it hasn't done it yet). So I figured I had better do it myself. One thing led to another, and this program now supports zoom windows on the old 64K ROM.

THE PATCH

If all I did was insert the WDEF, I'd get a nice looking window with some funny behavior (try it! nothing bad will happen). I need to make the Window Manager respond appropriately to mouse downs in a grow bracket.

So taking the hint from Melton and Ritter, I replaced the GrowWindow and SizeWindow routines. And while I was at it, I made my new "sizethewindow" routine a replacement for ZoomWindow as well, and added a replacement for TrackBox, so I could zoom windows as well as grow them.

BLOW-BY-BLOW

The program is contained in the file "Corners.p", the replacement routines in the unit "Patches.p", and the WDEF in the file "WDEF.p" (Okay, so I lack imagination!). There's another unit "Common.p" that at one time was substantial but now has only some minor type and constant declarations. The resource source file is "Corners.r".

File Edit

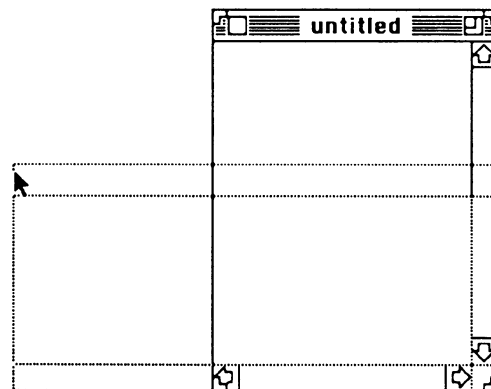


Figure 1. Growing The Window

When the user clicks in one of the corner brackets, the program calls FindWindow, which in turn calls the WDEF. The WDEF returns wInGrow if the click was close to any of the corners. Simple so far. FindWindow tells the program the click was in the grow box, and the program calls the Patches routine "growthewindow" in place of the ROM's GrowWindow.

Growthewindow has a lot of work to do. It must track the mouse, and draw a gray outline of the re-sized window as it does so. When the mouse is released, it must return not just the new dimensions of the window but the new location as well.

Growthewindow doesn't actually draw the gray outline; that is done by the WDEF. But it must tell the WDEF the rectangle to use when drawing. The basic method of computing this rectangle is (1) find the fixed corner of the window (the one opposite the corner being dragged), which I call the "pivot"; (2) track the mouse in a loop, each time getting the current mouse position; (3) use the wonderful Toolbox routine Pt2Rect to convert the mouse position and the pivot into a rectangle; and (4) call the WDEF to draw the gray outline from that rectangle.

That sounds simple but there are a lot of small things that cloud the picture. For example, the old gray outline must be erased before the new is drawn. And the mouse will rarely start out at the corner of the portrect, so a correction must be applied. And then there are various constraints on the rectangle, e.g., those imposed by the SizeRect parameter. Also, the rectangle must be in global coordinates.

When all this is done, and the user releases the mouse button, the routine must return the window's new size and location. What could be easier than just returning the last rectangle?

Now we're back to the Corners program. It has the new portrect, in global coordinates, as returned by growthewindow.

It calls another Patches routine, “sizethewindow”, which makes the actual changes in the window.

Sizethewindow does more than just re-size the window. Since the top left corner may have moved, the window must be moved as well. My first version of this routine called MoveWindow and SizeWindow but this resulted in two updates. Nesting the calls within ShowHide calls solved that but... I wanted something that would look just as smooth as the normal grow. ZoomWindow worked but limited the program to machines with the new ROM. Finally I discovered the QD routine MovePortTo, which does not update the screen! It is amazing what you can find, overlooked for years, in the QD chapter!

When I made that discovery, I realized that my sizethewindow routine could easily pinch-hit for ZoomWindow, giving me zoom windows on the old ROM. So for our next adventure, we will ask the user to click on the zoom box...

BLOWS AGAINST THE ZOOM BOX

Once again, the program calls FindWindow, which calls the WDEF, which returns wInZoomIn (I might note here that the WDEF could return anything I want, since this is all done privately). The program then calls the Patches routine “trackthebox”.

Trackthebox is kinda rough; I think it could be slimmed down but why bother? The time it takes is entirely determined by the user. Anyway, it goes into a loop, getting the mouse location, calling the WDEF to find out where that is, and calling the WDEF to highlight the zoom box if the mouse had just moved either in or out. When the button is released, the routine returns true if the mouse is still in the zoom box, false otherwise.

Now the program gets busy. First, note that one field of the record whose handle is stored in the refcon field of the window record is called “zoomrect”. Suggestive, huh? It gets the portrect of the window and compares it with a full screen portrect (this is done in global coordinates, of course). If the window is not at full screen, then its current portrect is saved in the zoomrect field, and sizethewindow zooms it to full screen. If it is at full screen, then the zoomrect is used to size it back down.

PUTTING IT TOGETHER

To add four-cornered windows to your program, add the Patches unit to your uses clause and the WDEF to the resource file. Then change the type of your windows from zoomDocProc or whatever to 800. Finally, change calls to GrowWindow and SizeWindow to “growthewindow” and “sizethewindow” (this will require more than just a name change but not much more).

If you want to support zooming, you’ll have to hack a little further. Let the description above be your guide.

Listing: Common.p

```
(*****)
unit Common;
(*****)
interface
(*****)
    key codes:
(*****)
```

const

```
enterkey      = 3;
backspace     = 8;
tabkey        = 9;
returnkey     = 13;
clearkey      = 27;
leftarrow     = 28;
rightarrow    = 29;
uparrow       = 30;
downarrow     = 31;
periodkey     = 46;
```

```
(*****
Dialog items:
*****
    themask      = 3;
(*****
Low-memory globals:
*****)
```

```
applscratch   = $A78;
bootdrive     = $210;
curappname    = $910;
curdirstore   = $398;
currenta5     = $904;
findername    = $2E0;
fsfcbien      = $3F6;
grayrgn       = $9EE;
iaznotify     = $33C;
mbarheight    = $BAA;
menuflash     = $A24;
resload       = $A5E;
rom85         = $28E;
sfsavedisk    = $214;
sysmap        = $A58;
windowlist    = $9D6;
```

```
(*****
Standard types:
*****)
type
    logical      = boolean;
    long         = longint;

    shortpointer = ^integer;
    longpointer  = ^long;
(*****
end.
*****)
```

Listing: Corners.p

```
(*****
Corners.p

Shell program for four-cornered windows.

(c) 1988, by Clifford Story & Attic Software

*****
program Corners;
(*****
    uses memtypes, quickdraw, osintf, toolintf, Common,
    Patches;
(*****
    Program constants:
*****)

const
    applenum    = 1001;
```

```

aboutitem = 1;
atticitem = 2;

filenum = 1002;
newitem = 1;
closeitem = 2;
quititem = 4;

editnum = 1003;
undoitem = 1;
cutitem = 3;
copyitem = 4;
pasteitem = 5;
clearitem = 6;

windownum = 1001;
scrollnum = 1001;

hoffset = 32;
voffset = 20;

(*****
Program types:
*****)
type
    wrecord = record
        window : WindowPtr;
        zoomrect : Rect;
        hscroll : ControlHandle;
        vscroll : ControlHandle;
        disprect : Rect;
    end;
    wpointer = ^wrecord;
    whandle = ^wpointer;

(*****
Program variables:
*****)
var
    APPLEMENU : MenuHandle;
    FILEMENU : MenuHandle;
    EDITMENU : MenuHandle;

    MOUSEREGION : RgnHandle;
    SCREENREGION : RgnHandle;

    MENUHEIGHT : integer;
    DRAGRECT : Rect;
    GROWRECT : Rect;
    SCREENRECT : Rect;

    COLUMNS : integer;
    ROWS : integer;
    WINDOWCOUNT : integer;

    DONE : logical;
    JEVENT : logical;
    MAINEVENT : EventRecord;

(*****
procedure _datainit; external;
*****)
($R-)
($SC+)

(*****
procedure panic;
begin
    ExitToShell;
end;
*****)

```

```

procedure initmac;
begin
    MaxApplZone;
    InitGraf(@thePort);
    InitFonts;
    InitWindows;
    InitCursor;
    InitMenus;
    TEInit;
    InitDialogs(@panic);

    UnloadSeg(@_datainit);
end;

(*****
procedure setupmenus;
begin
    APPLEMENU := GetMenu(applenum);
    AddResMenu(APPLEMENU, 'DRVR');
    InsertMenu(APPLEMENU, 0);

    FILEMENU := GetMenu(filenum);
    InsertMenu(FILEMENU, 0);

    EDITMENU := GetMenu(editnum);
    InsertMenu(EDITMENU, 0);

    DrawMenuBar;
end;
*****)

(*****
initglobals
*****)
Here's where I initialize all the global variables.
DRAGRECT is an argument to DragWindow; GROWRECT is an
argument to "growthewindow" (in the Patches unit);
SCREENRECT is used in zooming (it's the portrect of a full-
screen window); and ROWS and COLUMNS are used to stack the
windows on the screen.
(*****
procedure initglobals;
var
    index : integer;
    theshort : shortpointer;
begin
    for index := 1 to 10 do
        MoreMasters;

    if BitTst(Ptr(rom85), 0) then begin
        MENUHEIGHT := 20;
        JEVENT := false;
    end else begin
        theshort := shortpointer(mbarheight);
        MENUHEIGHT := theshort^;
        JEVENT := (NGetTrapAddress($A860, ToolTrap)
        <> NGetTrapAddress($A89F, ToolTrap));
    end;

    with screenBits.bounds do begin
        SetRect(DRAGRECT, left + 5, top + MENUHEIGHT + 5,
            right - 5, bottom - 25);
        SetRect(GROWRECT, 160, 100, right - left - 10,
            bottom - top - MENUHEIGHT - 10);
        SetRect(SCREENRECT, left + 5, top + MENUHEIGHT + 25,
            right - 5, bottom - 5);
        COLUMNS := 1 + ((right - left - 330) div hoffset);
        ROWS := 1 + ((bottom - top - MENUHEIGHT - 230) div
voffset);
        WINDOWCOUNT := 0;
    end;

    DONE := false;
end;

```

```

(*****
clickapplenue

This may not be new to you but it is to me! Instead of
using an alert for the "About..." box, I'm using a picture.
First I open a new GrafPort, then draw the picture in its
center. Note that I must re-draw the windows after the user
dismisses the screen, using PaintBehind.
*****
)
procedure clickapplenue(theitem: integer);
var
  itemname      : Str255;
  savedport     : GrafPtr;
  dummy         : integer;
  newport       : GrafPort;
  thepicture    : PicHandle;
  therect       : Rect;
begin
  if theitem > 3 then begin
    GetItem(APPLEMENU, theitem, itemname);
    GetPort(savedport);
    dummy := OpenDeskAcc(itemname);
    SetPort(savedport);
  end else if theitem < 3 then begin

    InitCursor;
    GetPort(savedport);
    OpenPort(@newport);
    SetPort(@newport);

    thepicture := PicHandle(GetResource('PICT',
      1000 + theitem));
    with thepicture^^.picFrame do
      SetRect(therect, 0, 0, right - left, bottom - top);
    with screenBits.bounds, therect.botright do
      OffsetRect(therect, (right - left - h) div 2,
        (bottom - top - v) div 3);
    DrawPicture(thepicture, therect);

    repeat until Button;

    ClosePort(@newport);
    EnableItem(EDITMENU, 0);
    DrawMenuBar;
    PaintBehind(WindowPeek(FrontWindow),
      RgnHandle(longpointer(grayrgn)^^));

    SetPort(savedport);
    FlushEvents(everyEvent, 0);

  end;
end;

(*****
placewindow

This routine cost me $40.00 (it's the only thing I got out
of "Macintosh Revealed").
*****
)
procedure placewindow(thewindow : WindowPtr);
var
  left      : integer;
  top       : integer;
begin
  left := 5 + hoffset * (WINDOWCOUNT mod COLUMNS);
  top := 5 + MENUHEIGHT
    + voffset * (1 + (WINDOWCOUNT mod ROWS));

  MoveWindow(thewindow, left, top, true);
end;

(*****
procedure wsize(thewindow : whandle);

```

```

var
  thewidth   : integer;
  theheight  : integer;
begin
  HLock(Handle(thewindow));
  with thewindow^^ do begin

    thewidth := window^.portRect.right;
    theheight := window^.portRect.bottom;

    HideControl(hscroll);
    HideControl(vscroll);

    MoveControl(vscroll, thewidth - 15, -1);
    SizeControl(vscroll, 16, theheight - 13);

    MoveControl(hscroll, 15, theheight - 15);
    SizeControl(hscroll, thewidth - 29, 16);

    disprect := window^.portRect;
    InsetRect(disprect, 10, 10);
    OffsetRect(disprect, -8, -8);

    ShowControl(hscroll);
    ShowControl(vscroll);

  end;
  HUnlock(Handle(thewindow));
end;

(*****
procedure donew;
label
  100;
var
  thehandle   : whandle;
  thewindow   : WindowPtr;
begin
  thehandle := whandle(NewHandle(sizeof(wrecord)));
  if thehandle = nil then
    goto 100;

  HLock(Handle(thehandle));
  with thehandle^^ do begin

    window := GetNewWindow(windownum, nil, WindowPtr(-
1));

    if window = nil then begin
      DisposHandle(Handle(thehandle));
      goto 100;
    end;
    SetWRefCon(window, long(thehandle));

    zoomrect := SCREENRECT;

    hscroll := GetNewControl(scrollnum, window);
    if hscroll = nil then begin
      DisposeWindow(window);
      DisposHandle(Handle(thehandle));
      goto 100;
    end;

    vscroll := GetNewControl(scrollnum, window);
    if vscroll = nil then begin
      DisposeControl(hscroll);
      DisposeWindow(window);
      DisposHandle(Handle(thehandle));
      goto 100;
    end;

    placewindow(window);
    wsize(thehandle);
    ShowWindow(window);

```

```

WINDOWCOUNT := WINDOWCOUNT + 1;

end;
HUnlock(Handle(thehandle));
100: end;

(*****
procedure doclose(thepeek : WindowPeek);
var
  thehandle      : whandle;
begin
  if thepeek^.windowkind < 0 then
    CloseDeskAcc(thepeek^.windowkind)
  else begin
    thehandle := whandle(thepeek^.refCon);
    DisposeControl(thehandle^.hscroll);
    DisposeControl(thehandle^.vscroll);
    DisposeWindow(WindowPtr(thepeek));
    DisposHandle(Handle(thehandle));
    WINDOWCOUNT := WINDOWCOUNT - 1;
  end;
end;

(*****
procedure clickfilenmenu(itemchoice : integer);
var
  dummy      : logical;
begin
  case itemchoice of
    newitem   : donew;
    closeitem : doclose(WindowPeek(FrontWindow));
    quititem  : DONE := true;
  end;
end;

(*****
procedure checkmenu(thewindow : WindowPeek);
begin
  DisableItem(EDITMENU, 0);

  if thewindow = nil then
    DisableItem(FILEMENU, closeitem)
  else begin
    if thewindow^.windowkind <> userKind then
      EnableItem(EDITMENU, 0);
      EnableItem(FILEMENU, closeitem);
    end;
  end;

(*****
procedure clickinmenu;
var
  choice      : long;
begin
  checkmenu(WindowPeek(FrontWindow));
  choice := MenuSelect(MAINEVENT.where);

  case HiWord(choice) of
    applenum  : clickapplenmenu(LoWord(choice));
    filenum   : clickfilenmenu(LoWord(choice));
    editnum   : if not SystemEdit(LoWord(choice) - 1)
then;
  end;

  HiliteMenu(0);
end;

(*****
clickingrow

This is just like the usual grow routine, except that I'm
calling the Patches routines "growthewindow" and
"sizethewindow". These behave like GrowWindow and
SizeWindow, but in the four-cornered context.

```

```

(*****
procedure clickingrow(thewindow : WindowPtr);
var
  newrect      : Rect;
begin
  if growthewindow(thewindow, MAINEVENT.where,
    GROWRECT, newrect) then begin
    sizethewindow(thewindow, newrect);
    wsize(whandle(GetWRefCon(thewindow)));
  end;
end;

(*****
procedure clickingoaway(thewindow : WindowPtr);
begin
  if TrackGoAway(thewindow, MAINEVENT.where) then
    doclose(WindowPeek(thewindow));
  end;

(*****
clickinzoom

Custom zooming - this will work on any ROM. "trackthebox"
performs exactly like TrackBox; I wrote it so I could track
clicks in the zoom box regardless of ROM version. The
actual zooming uses "sizethewindow", which I already have
for growing windows.

Each window has a field called "zoomrect". If I'm zooming
to full screen, I save the old portRect in zoomrect first.
If I'm already at full screen and I'm zooming back, I use
the save zoomrect as the target portRect. Note, though,
that zoomrect must be in global coordinates.

(*****
procedure clickinzoom(thewindow : WindowPtr);
var
  thehandle     : whandle;
  therect       : Rect;
begin
  if trackthebox(thewindow) then begin

    thehandle := whandle(GetWRefCon(thewindow));
    therect := thewindow^.portRect;
    EraseRect(therect);

    LocalToGlobal(therect.topleft);
    LocalToGlobal(therect.botright);

    if EqualRect(therect, SCREENRECT) then
      therect := thehandle^.zoomrect
    else begin
      thehandle^.zoomrect := therect;
      therect := SCREENRECT;
    end;

    sizethewindow(thewindow, therect);
    wsize(thewindow);
  end;
end;

(*****
procedure aclick;
var
  location      : integer;
  thewindow     : WindowPtr;
begin
  location := FindWindow(MAINEVENT.where, thewindow);

  case location of
    inDesk      : SysBeep(1);
    inMenuBar   : clickinmenu;
    inSysWindow : SystemClick(MAINEVENT, thewindow);
    inContent   : SelectWindow(thewindow);
  end;
end;

```



```

inDrag : DragWindow(thewindow, MAINEVENT.where,
DRAGRECT);
inGrow : clickingrow(thewindow);
inGoAway : clickingoaway(thewindow);
inZoomIn : clickinzoom(thewindow);
end;
end;
(*****
procedure akey;
var
  charcode : integer;
  choice : long;
begin
  if BitAnd(MAINEVENT.modifiers, cmdKey) <> 0 then begin
    charcode := BitAnd(MAINEVENT.message, charCodeMask);
    checkmenu(WindowPeek(FrontWindow));
    choice := MenuKey(chr(charcode));

    if choice <> 0 then begin
      case HiWord(choice) of
        applenum : clickapplenum(LoWord(choice));
        filenum : clickfilemenu(LoWord(choice));
        editnum : if not SystemEdit(LoWord(choice)) -
1) then;
      end;
      HiliteMenu(0);
    end;
  end;
end;
end;
(*****
procedure anactivate(thewindow : WindowPtr);
begin
  SetPort(thewindow);
  DrawGrowIcon(thewindow);
end;
(*****
procedure anupdate(thewindow : WindowPtr);
var
  savedport : GrafPtr;
begin
  GetPort(savedport);
  SetPort(thewindow);

  BeginUpdate(thewindow);

  ClipRect(thewindow^.portRect);
  EraseRect(thewindow^.portRect);
  DrawGrowIcon(thewindow);
  DrawControls(thewindow);

  EndUpdate(thewindow);

  SetPort(savedport);
end;
(*****
procedure mainloop;
var
  dummy : logical;
begin
  repeat
    if JEVENT then
      dummy := waitnextevent(everyEvent, MAINEVENT,
        GetCaretTime, nil)
    else begin
      SystemTask;
      dummy := GetNextEvent(everyEvent, MAINEVENT);
    end;

```

```

if dummy then begin
  case MAINEVENT.what of
    mouseDown : aclick;
    keyDown : akey;
    autoKey : akey;
    activateEvt :
      anactivate(WindowPtr(MAINEVENT.message));
    updateEvt :
      anupdate(WindowPtr(MAINEVENT.message));
  end;
end;

until DONE;
end;
(*****
procedure shutdown;
var
  thelong : longpointer;
  thepeek : WindowPeek;
begin
  thelong := longpointer(windowlist);
  thepeek := WindowPeek(thelong^);

  while thepeek <> nil do begin
    doclose(thepeek);
    thepeek := thepeek^.nextwindow;
  end;

end;
(*****
begin
  initmac;
  setupmenus;
  initglobals;
  mainloop;
  shutdown;
end.
(*****

```

Listing: Patches.p

```

(*****
Patches.p
ROM replacements for four-cornered windows.
(c) 1988, by Clifford Story & Attic Software
*****
unit Patches;
(*****
interface
(*****
uses memtypes, quickdraw, osintf, toolintf, Common;
(*****
function growthewindow(theWindow : WindowPtr; startpoint :
Point; sizerect : Rect; var newportrect : Rect) : logical;

procedure sizethewindow(theWindow : WindowPtr; newrect :
Rect);

function trackthebox(theWindow : WindowPtr) : logical;
(*****
implementation
(*****
($R-)
($SC+)
(*****
callWDEF

```

The inline code here is:

```
movea.l    (SP)+,A0
jsr        A0
```

That is, it pops the "address" argument into register A0, and then calls the routine at that address.

```
function callWDEF(variation : integer; thewindow : WindowPtr; message : integer; param : Ptr; address : Ptr) : long;
inline $205F, $4E90;
```

growthewindow

This routine is an analog of the Toolbox trap GrowWindow; it follows mouse with a gray outline of the window's new size.

I'm going to draw in Window Manager port, and I'm going to set the clipping to full screen (the low-memory global "grayrgn" contains a handle to the region consisting of everything but the menu bar). Since I don't want to upset the Window Manager, I'll save the old clip region first, so I can restore it at the end. The pen mode makes drawing its own inverse. This routine draws by calling the WDEF with a "wGrow" message. All this is in the Window Manager section of Inside Mac; I'm just approaching it from the other side.

The mouse is down in one corner of the window. The diagonally opposite window is fixed for this operation; I call it the "pivot". The mouse is probably not on the corner exactly; I use "offset" to correct the mouse position. And this corrected position will be corrected further to bring it within "limitrect" if it strays outside.

From here on, it's simple. Draw the first outline; then, so long as the mouse stays down, repeatedly erase it (by redrawing) and draw the new outline. When the mouse is released, erase the outline and return the new portRect.

```
function growthewindow(thewindow : WindowPtr; startpoint : Point; sizerect : Rect; var newportrect : Rect) : logical;
```

var

```
savedport      : GrafPtr;
drawport       : GrafPtr;
savedclip      : RgnHandle;
thepointer     : longpointer;
WDEF           : Handle;
newrect        : Rect;
west           : logical;
north          : logical;
pivot          : Point;
offset         : Point;
limitrect      : Rect;
menuheight     : integer;
theshort       : shortpointer;
dummy          : long;
oldrect        : Rect;
newpoint       : Point;
```

begin

```
GetPort(savedport);
GetWMgrPort(drawport);
SetPort(drawport);
```

```
savedclip := NewRgn;
GetClip(savedclip);
```

```
thepointer := longpointer(grayrgn);
SetClip(RgnHandle(thepointer^));
```

```
PenMode(notPatXor);
PenPat(gray);
```

```
WDEF := WindowPeek(thewindow)^.windowdefproc;
HLock(WDEF);
```

```
newrect := thewindow^.portRect;
with thewindow^.portBits.bounds do
  OffsetRect(newrect, -left, -top);
```

with newrect do begin

```
if startpoint.v <= ((top + bottom) div 2) then begin
  pivot.v := bottom;
  offset.v := top - startpoint.v;
  limitrect.top := bottom - sizerect.bottom;
  limitrect.bottom := bottom - sizerect.top;
end else begin
  pivot.v := top;
  offset.v := bottom - startpoint.v;
  limitrect.top := top + sizerect.top;
  limitrect.bottom := top + sizerect.bottom;
end;
```

```
if startpoint.h <= ((left + right) div 2) then begin
  pivot.h := right;
  offset.h := left - startpoint.h;
  limitrect.left := right - sizerect.right;
  limitrect.right := right - sizerect.left;
end else begin
  pivot.h := left;
  offset.h := right - startpoint.h;
  limitrect.left := left + sizerect.left;
  limitrect.right := left + sizerect.right;
end;
```

end;

```
if BitTst(Ptr(rom85), 0) then
  menuheight := 38
```

else begin

```
theshort := shortpointer(mbarheight);
menuheight := theshort^ + 18;
```

end;

```
if limitrect.top < menuheight then
  limitrect.top := menuheight;
```

```
newpoint := startpoint;
AddPt(offset, newpoint);
```

```
newpoint := Point(PinRect(limitrect, newpoint));
Pt2Rect(pivot, newpoint, newrect);
```

```
dummy := callWDEF(0, thewindow, wGrow, @newrect, WDEF^);
oldrect := newrect;
```

while StillDown do begin

```
GetMouse(newpoint);
AddPt(offset, newpoint);
```

```
newpoint := Point(PinRect(limitrect, newpoint));
Pt2Rect(pivot, newpoint, newrect);
```

```
if not EqualRect(newrect, oldrect) then begin
  dummy := callWDEF(0, thewindow, wGrow, @oldrect,
```

```
WDEF^);
  dummy := callWDEF(0, thewindow, wGrow, @newrect,
```

```
WDEF^);
  oldrect := newrect;
```

end;

end;

```
dummy := callWDEF(0, thewindow, wGrow, @oldrect, WDEF^);
```

```

HUnlock(WDEF);

PenNormal;
SetClip(savedclip);
DisposeRgn(savedclip);
SetPort(thewindow);

oldrect := thewindow^.portRect;
LocalToGlobal(oldrect.topleft);
LocalToGlobal(oldrect.botrigh);

if EqualRect(oldrect, newrect) then
  growthewindow := false
else begin
  newportrect := newrect;
  growthewindow := true;
end;

SetPort(savedport);
end;

(*****
sizethewindow

This routine is an analog of the Toolbox trap SizeWindow.
It just precedes SizeWindow with MovePortTo, which moves the
window without updating the screen. After this, I update
the Window Manager port's clipping, so it will draw the
window correctly.
*****)
procedure sizethewindow(thewindow : WindowPtr; newrect :
Rect);
var
  drawport      : GrafPtr;
begin
  EraseRect(thewindow^.portRect);

  with newrect do begin
    MovePortTo(left, top);
    SizeWindow(thewindow, right - left, bottom - top,
false);
  end;

  InvalRect(thewindow^.portRect);

  GetWMgrPort(drawport);
  SetPort(drawport);
  SetClip(WindowPeek(thewindow)^.strucrgn);

  SetPort(thewindow);
  ClipAbove(WindowPeek(thewindow));
end;

(*****
trackthebox

This routine is a duplicate of TrackBox; I would use
TrackBox but it isn't available in the old ROM.

The idea is to highlight the zoom box, and then track the
mouse while the button remains down. The zoom box should be
highlighted when the mouse is in it, and unhighlighted when
it is not. When the button is finally released, the routine
returns true if the mouse is in the zoom box, and false if
it is not.
*****)
function trackthebox(thewindow : WindowPtr) : logical;
var
  highlight      : logical;
  savedport      : GrafPtr;
  drawport       : GrafPtr;
  thepointer     : longpointer;
  WDEF           : Handle;

```

```

  thepoint       : Point;
  thepart        : long;
begin
  GetPort(savedport);
  GetWMgrPort(drawport);
  SetPort(drawport);

  WDEF := WindowPeek(thewindow)^.windowdefproc;
  HLock(WDEF);

  thepart := callWDEF(0, thewindow,
    wDraw, Ptr(wInZoomIn), WDEF^);
  highlight := true;

  while StillDown do begin
    GetMouse(thepoint);
    thepart := callWDEF(0, thewindow,
      wHit, Ptr(thepoint), WDEF^);

    if not highlight and (thepart = wInZoomIn) then
begin
      thepart := callWDEF(0, thewindow,
        wDraw, Ptr(wInZoomIn), WDEF^);
      highlight := true;
    end else if highlight and (thepart <> wInZoomIn)
then begin
      thepart := callWDEF(0, thewindow,
        wDraw, Ptr(wInZoomIn), WDEF^);
      highlight := false;
    end;

  end;

  if highlight then
    thepart := callWDEF(0, thewindow,
      wDraw, Ptr(wInZoomIn), WDEF^);

  HUnlock(WDEF);

  SetPort(savedport);

  trackthebox := highlight;
end;
(*****
end.
*****)

Listing: WDEF.p

(*****
WDEF.p
WDEF for four-cornered window.
(c) 1988, by Clifford Story & Attic Software
*****)
unit WDEF;
(*****
interface
(*****
uses mentypes, quickdraw, osintf, toolintf, Common;
(*****
function windowdef(variation : integer; thewindow : Win-
dowPeek; message : integer; param : long) : long;
(*****
implementation
(*****
($R-)
($SC+)
(*****
datahandle

```

This is horribly inefficient; each window saves the data for each BitMap in its datahandle. So each window duplicates this information (which must also be in the code, or where

would it come from?). I've got an assembler version of this WDEF that eliminates all this duplicate data; you could do it in Pascal by moving the StuffHex's out of the init routine and into the draw routine.

*****)

type

```
rectrecord = record
  dragrect : Rect;
  closerect : Rect;
  zoomrect : Rect;
  NWmap : BitMap;
  NWdata : array [1..28] of integer;
  NEmap : BitMap;
  NEdata : array [1..28] of integer;
  SWmap : BitMap;
  SWdata : array [1..8] of integer;
  SEmap : BitMap;
  SEdata : array [1..8] of integer;
  erasedata : array [1..8] of integer;
  closerect : BitMap;
  closedata : array [1..9] of integer;
  zoommap : BitMap;
  zoomdata : array [1..9] of integer;
end;
```

```
rectpointer = ^rectrecord;
recthandle = ^rectpointer;
```

```
QDrecord = record
  randseed : long;
  screenbits : bitmap;
  arrow : cursor;
  dkgray : pattern;
  ltgray : pattern;
  gray : pattern;
  black : pattern;
  white : pattern;
  theport : grafptr;
end;
QDpointer = ^QDrecord;
```

*****)

```
procedure wdodraw(thewindow : WindowPeek; param : long);
forward;
function wdohit(thewindow : WindowPeek; thepoint : Point) :
long; forward;
procedure wdocalc(thewindow : WindowPeek); forward;
procedure wdonev(thewindow : WindowPeek); forward;
procedure wdodispose(thewindow : WindowPeek); forward;
procedure wdogrow(therect : rectpointer); forward;
procedure wdoicon(thewindow : WindowPtr); forward;
*****)
```

```
function windowdef(variation : integer; thewindow : Win-
dowPeek; message : integer; param : long) : long;
begin
```

```
  windowdef := 0;
```

```
  case message of
```

```
    wDraw : wdodraw(thewindow, param);
    wHit : windowdef := wdohit(thewindow,
Point(param));
    wCalcRgns : wdocalc(thewindow);
    wNew : wdonev(thewindow);
    wDispose : wdodispose(thewindow);
    wGrow : wdogrow(rectpointer(param));
    wDrawGIcon : wdoicon(WindowPtr(thewindow));
```

```
  end;
```

```
end;
```

*****)

QDglobals

A WDEF can't use globals, hence cannot reference the

Quickdraw globals directly. This routine returns a pointer to the Quickdraw globals, which the WDEF can use instead.

*****)

function QDglobals : QDpointer;

var

thepointer : longpointer;

begin

```
  thepointer := longpointer(currenta5);
  thepointer := longpointer(thepointer^);
  QDglobals := QDpointer(long(thepointer^)
    - sizeof(QDrecord) + sizeof(grafptr));
```

end;

*****)

wdodraw

This routine draws the window. There are two special cases: if the low word of the "param" argument is wInGoAway, then this is a call for close box highlighting; if it is wInZoomIn, then this is a call for zoom box highlighting.

I draw the various parts of the window in order. First, I recalculate the various regions. Then I draw the window's outline and shadow. Next is the title bar: outline the title bar, then erase its interior. Draw the title (truncated, if necessary, by clipping), then corners with CopyBits, and finally the horizontal lines between corners and the title. Last, draw the lower corners. Note that the corner brackets are drawn only if the window is highlighted. *****)

procedure wdodraw(thewindow : WindowPeek; param : long);

var

```
therect : Rect;
width : integer;
limitrect : Rect;
savedclip : RgnHandle;
height : integer;
index : integer;
```

begin

```
  with thewindow^, QDglobals^.thePort^ do begin
```

```
    if visible then begin
```

```
      HLock(dataHandle);
      with recthandle(dataHandle)^ do begin
```

```
        if LoWord(param) = wInGoAway then begin
          closerect.baseAddr := @closedata;
          CopyBits(closerect, portBits, closerect.bounds,
            closerect.bounds, srcXor, nil);
        end else if LoWord(param) = wInZoomIn then
```

begin

```
          zoommap.baseAddr := @zoomdata;
          CopyBits(zoommap, portBits, zoommap.bounds,
            zoommap.bounds, srcXor, nil);
        end else if param = 0 then begin
```

```
          wdocalc(thewindow);
```

```
          FrameRgn(strucrgn);
          therect := strucrgn^.rgnBBox;
```

```
          with therect do begin
```

```
            MoveTo(left, bottom - 2);
            LineTo(right - 2, bottom - 2);
            LineTo(right - 2, top);
```

```
          end;
```

```
          therect := dragrect;
          InsetRect(therect, 1, 1);
          EraseRect(therect);
          FrameRect(dragrect);
```

```

with dragrect do
    width := (right + left - titlewidth) div 2;
    if width < NWmap.bounds.right then
        width := NWmap.bounds.right + 3;

    limitrect.left := NWmap.bounds.right;
    limitrect.top := NWmap.bounds.top - 2;
    limitrect.right := NEmap.bounds.left - 3;
    limitrect.bottom := NWmap.bounds.bottom + 2;

    savedclip := NewRgn;
    GetClip(savedclip);
    ClipRect(limitrect);
    SectRgn(savedclip, clipRgn, clipRgn);

    MoveTo(width, dragrect.bottom - 5);
    DrawString(titlehandle^^);

    SetClip(savedclip);
    DisposeRgn(savedclip);

    if hilited then begin

        NWmap.baseAddr := @NWdata;
        CopyBits(NWmap, portBits, NWmap.bounds,
            NWmap.bounds, srcCopy, nil);

        NEmap.baseAddr := @NEdata;
        CopyBits(NEmap, portBits, NEmap.bounds,
            NEmap.bounds, srcCopy, nil);

        if (width - 7) > NWmap.bounds.right then

            height := closerect.top;
            for index := 1 to 6 do begin
                MoveTo(NWmap.bounds.right, height);
                LineTo(width - 7, height);
                Move(titlewidth + 13, 0);
                LineTo(NEmap.bounds.left - 1, height);
                height := height + 2;
            end;
        end;

    end;

    if hilited then begin
        SEmap.baseAddr := @SEdata;
        SWmap.baseAddr := @SWdata;
    end else begin
        SWmap.baseAddr := @erasedata;
        SEmap.baseAddr := @erasedata;
    end;

    CopyBits(SWmap, portBits, SWmap.bounds,
        SWmap.bounds, srcCopy, nil);
    CopyBits(SEmap, portBits, SEmap.bounds,
        SEmap.bounds, srcCopy, nil);

end;
end;
HUnlock(dataHandle);
end;
end;
end;

```

```

(*****
wdohit

```

This routine hit-tests a point, and returns window part that the point lies in. The first section tests for the grow brackets; if the point is near the top, and also near the left edge, then it's in top-left bracket, ect. Note that unhighlighted windows don't have grow brackets. The rest is

```

just testing for specific rectangles.
*****
function wdohit(thewindow : WindowPeek; thepoint : Point) :
long;
    label
        100, 200;
    var
        therect      : Rect;
    begin

        wdocalc(thewindow);

        with thewindow^ do begin

            HLock(dataHandle);
            with recthandle(dataHandle)^ do begin

                wdohit := wNoHit;

                if hilited then begin
                    therect := structrgn^^.rgnBBox;
                    if (thepoint.v > (therect.top + 8))
                        and (thepoint.v < (therect.bottom - 8))
                then
                    goto 100;
                    if (thepoint.h > (therect.left + 8))
                        and (thepoint.h < (therect.right - 8))
                then
                    goto 100;
                    wdohit := wInGrow;
                    goto 200;
                end;

                100: if PtInRect(thepoint, closerect) and hilited and
                    goAwayFlag then
                    wdohit := wInGoAway
                else if PtInRect(thepoint, zoomrect) and hilited
                then
                    wdohit := wInZoomIn
                else if PtInRect(thepoint, contrgn^^.rgnBBox)
                then
                    wdohit := wInContent
                else if PtInRect(thepoint, dragrect) then
                    wdohit := wInDrag;

                end;
                HUnlock(dataHandle);

            end;

        200: end;

    (*****
    wdocalc
    -----
    This routine calculates all regions and rectangles in the
    window. Note these are all in local coordinates, and that
    I assume that window's portBits.bounds.topleft is the point
    (0, 0).
    *****
    procedure wdocalc(thewindow : WindowPeek);
    var
        therect      : Rect;
    begin

        with thewindow^ do begin

            therect := port.portRect;
            with port.portBits.bounds do
                OffsetRect(therect, - left, - top);

            OpenRgn;
            MoveTo(therect.left + 1, therect.top);
            LineTo(therect.right - 1, therect.top);

```



```

    MoveTo(dragrect.left, dragrect.top - 18);
    LineTo(dragrect.right, dragrect.top - 18);
    LineTo(dragrect.right, dragrect.bottom);
    LineTo(dragrect.left, dragrect.bottom);
    LineTo(dragrect.left, dragrect.top - 18);

    MoveTo(dragrect.left, dragrect.top);
    LineTo(dragrect.right, dragrect.top);

end;

end;

(*****
wdoicon
-----
This routine is supposed to draw the grow icon. This window
doesn't have a grow icon, so it only draws the outline for
scroll bars.
*****)
procedure wdoicon(thewindow : WindowPtr);
var
    savedport : GrafPtr;
begin
    GetPort(savedport);
    SetPort(thewindow);

    with thewindow^.portRect do begin
        MoveTo(left, bottom - 15);
        LineTo(right, bottom - 15);
        MoveTo(right - 15, top);
        LineTo(right - 15, bottom);
    end;

    SetPort(savedport);

end;

(*****
end.
*****)

Listing: Corners.R

/*****
Corners.r
Resources for four-cornered window.
(c) 1988, by Clifford Story & Attic Software
*****/
#include "types.r"
/*****
Menu resources
*****/
resource 'MENU' (1001, preload) (
    1001,
    textMenuProc,
    0x7FFFFFFB,
    enabled,
    apple,
    ( /* array: 3 elements */
        /* [1] */
        "About Corners...", 1, "", "", plain,
        /* [2] */
        "About Attic Software...", noicon, "", "", plain,
        /* [3] */
        "-", noIcon, "", "", plain
    )
);

resource 'MENU' (1002, preload) (
    1002,
    textMenuProc,
    0x7FFFFFF9,

```

```

    enabled,
    "File",
    ( /* array: 4 elements */
        /* [1] */
        "New", noIcon, "N", "", plain,
        /* [2] */
        "Close", noIcon, "K", "", plain,
        /* [3] */
        "-", noIcon, "", "", plain,
        /* [4] */
        "Quit", noIcon, "Q", "", plain
    )
);

```

```

resource 'MENU' (1003, preload) (
    1003,
    textMenuProc,
    0x7FFFFFFD,
    enabled,
    "Edit",
    ( /* array: 6 elements */
        /* [1] */
        "Undo", noIcon, "Z", "", plain,
        /* [2] */
        "-", noIcon, "", "", plain,
        /* [3] */
        "Cut", noIcon, "X", "", plain,
        /* [4] */
        "Copy", noIcon, "C", "", plain,
        /* [5] */
        "Paste", noIcon, "V", "", plain,
        /* [6] */
        "Clear", noIcon, "", "", plain,
    )
);

```

```

/*****
Picture resources
*****/
resource 'PICT' (1001, "About", purgeable) (
    1001,
    (7, 7, 308, 499),
    $"1101 A000 82A0 008C 0100 0A00 0700 0701"
    $"3401 F30A 0000 0000 0000 0000 0B00 1B00"
    $"1B44 0009 0009 0132 01F1 0700 0200 0248"
    $"A100 9600 0606 0000 0002 03A1 009A 0008"
    $"FFFD 0000 00D4 0000 A000 9803 0003 0D00"
    $"0A28 012A 0026 22A9 2031 3938 3820 6279"
    $"2043 6C69 6666 6F72 6420 5374 6F72 7920"
    $"616E 6420 4174 7469 6329 AC22 2053 6F66"
    $"7477 6172 652C 2050 2E4F 2E20 426F 7820"
    $"3231 392C 2047 6F6C 6574 612C 2043 29A6"
    $"1161 6C69 666F 726E 6961 2020 2039 3331"
    $"3136 A000 99A0 0097 A100 9600 0606 0000"
    $"0002 03A1 009A 0008 FFFA 0000 0027 0000"
    $"A000 9804 050D 0012 2800 2000 E007 436F"
    $"726E 6572 73A0 0099 A000 97A1 0096 0006"
    $"0500 0000 0203 A100 9A00 0800 4400 0000"
    $"DA00 00A0 0098 0400 0D00 0C28 0057 001E"
    $"2254 6869 7320 7072 6F67 7261 6D20 6465"
    $"6D6F 6E73 7472 6174 6573 2061 6E20 696E"
    $"7465 7229 E022 6573 7469 6E67 2065 7874"
    $"656E 7469 6F6E 206F 6620 7468 6520 4D61"
    $"6369 6E74 6F73 68D0 A000 99A1 009A 0008"
    $"0034 0000 00DA 0000 A000 9828 0067 001E"
    $"2269 6E74 6572 6661 6365 3A20 2077 696E"
    $"646F 7773 2074 6861 7420 6361 6E20 6265"
    $"2072 6529 D622 2D73 697A 6564 2066 726F"
    $"6D20 616E 7920 636F 726E 6572 2028 696E"
    $"7374 6561 6420 6F66 29D7 010D A000 99A1"
    $"009A 0008 0024 0000 00DA 0000 A000 9828"
    $"0077 001E 226F 6E6C 7920 6672 6F6D 2074"
    $"6865 206C 6F77 6572 2072 6967 6874 292E"
    $"2020 4561 6368 2029 CF22 636F 726E 6572"

```

```

$2068 6173 2061 20D2 6772 6F77 2062 7261"
$636B 6574 D320 7768 6963 680D A000 99A1"
$009A 0008 0014 0000 00DA 0000 A000 9828"
$0087 001E 2279 6F75 2063 616E 2064 7261"
$6720 6A75 7374 206C 696B 6520 7468 6520"
$7573 7561 6C20 6729 CF0A 726F 7720 6963"
$6F6E 2E0D A000 99A1 009A 0008 0004 0000"
$00DA 0000 A000 9828 0097 001E 010D A000"
$99A1 009A 0008 FFF4 0000 00DA 0000 A000"
$982A 1022 5769 6E64 6F77 7320 7769 7468"
$2074 6869 7320 6665 6174 7572 6520 776F"
$756C 6420 7072 29E1 1B6F 6261 626C 7920"
$6265 206D 6F73 7420 7573 6566 756C 2069"
$6E20 610D A000 99A1 009A 0008 FFE4 0000"
$00DA 0000 A000 9828 0087 001E 2267 7261"
$7068 6963 7320 7072 6F67 7261 6D2C 206C"
$6573 7320 7573 6566 756C 2069 6E20 6129"
$D420 2074 6578 7420 7072 6F67 7261 6D2E"
$2020 5468 6973 2070 6172 7469 6375 6C61"
$720D A000 99A1 009A 0008 FFD4 0000 00DA"
$0000 A000 9828 00C7 001E 2270 726F 6772"
$616D 2069 7320 6F66 206E 6F20 7573 6520"
$6174 2061 6C6C 3B20 6974 2773 2029 C622"
$6A75 7374 2061 2074 6561 7365 7220 666F"
$7220 7468 6520 436F 726E 6572 7320 6172"
$7469 29D0 0463 6C65 0DA0 0099 A100 9A00"
$08FF C400 0000 DA00 00A0 0098 2800 D700"
$1E22 696E 2074 6865 2046 6562 7275 6172"
$7920 3139 3839 2069 7373 7565 206F 6620"
$4061 6320 29D7 1F54 7574 6F72 2C20 7768"
$6963 6820 696E 636C 7564 6573 2074 6865"
$2066 756C 6C0D A000 99A1 009A 0008 FFB4"
$0000 00DA 0000 A000 9828 00E7 001E 1350"
$6173 6361 6C20 736F 6562 7063 6F64"
$652E A000 99A0 0097 A000 8DA0 0083 FF"

```

```

);
resource 'PICT' (1002, "Attic", purgeable) (
    1125,
    (7, 7, 306, 497),
    $1101 A000 82A0 008C 0100 0A00 0700 0701"
    $3201 F10A 0000 0000 0000 0000 0800 1800"
    $1B44 0009 0009 0130 01EF 0700 0200 0248"
    $A100 9600 0606 0000 0002 03A1 009A 0008"
    $FFFA 0000 004B 0000 A000 9803 0003 0405"
    $0D00 122B BA2B 0E41 7474 6963 2053 6F66"
    $7477 6172 65A0 0099 A000 97A1 0096 0006"
    $0500 0000 0203 A100 9A00 0800 5C00 0000"
    $E600 00A0 0098 0400 0D00 0C28 004D 001D"
    $2241 7474 6963 2053 6F66 7477 6172 6520"
    $6973 2061 2073 6D61 6C6C 204D 6163 696E"
    $746F 7329 DC22 6820 7072 6F67 7261 6D6D"
    $696E 6720 636F 6D70 616E 792E 2069 6E20"
    $6275 7369 6E65 7373 29E2 010D A000 99A1"
    $009A 0008 004C 0000 00E6 0000 A000 9828"
    $005D 001D 2273 696E 6365 2031 3938 362E"
    $2020 5765 2064 6F20 6120 7661 7269 6574"
    $7920 6F66 2077 6F29 D722 726B 3B20 D249"
    $6465 616C 696E 6572 D32C 2061 2073 6861"
    $7265 7761 7265 206F 7574 6C69 29D1 056E"
    $6572 2C0D A000 99A1 009A 0008 003C 0000"
    $00E6 0000 A000 9828 006D 001D 2269 7320"
    $6F75 7220 6265 7374 2D6B 6E6F 776E 2070"
    $726F 6475 6374 2E20 2057 6520 616C 7329"
    $D81B 6F20 646F 2063 6F6E 7472 6163 7420"
    $7072 6F67 7261 6D6D 696E 672E 0DA0 0099"
    $A100 9A00 0800 2C00 0000 E600 00A0 0098"
    $2800 7D00 1D01 0DA0 0099 A100 9A00 0800"
    $1C00 0000 E600 00A0 0098 2A10 1E57 6520"
    $6361 6E20 6265 2072 6561 6368 6564 2062"
    $7920 6D61 696C 2061 743A 0DA0 0099 A100"
    $9A00 0800 0C00 0000 E600 00A0 0098 2A10"
    $010D A000 99A1 009A 0008 FFFC 0000 00E6"
    $0000 A000 982A 1022 2020 2020 2020 2020"

```

```

$2020 2020 2020 2020 2020 2020 2020 2020"
$2020 2020 2020 2041 7474 2991 0C69 6320"
$536F 6674 7761 7265 0DA0 0099 A100 9A00"
$08FF EC00 0000 E600 00A0 0098 2800 BD00"
$1D22 2020 2020 2020 2020 2020 2020 2020"
$2020 2020 2020 2020 2020 2020 2020 2020"
$2050 2E4F 298F 0A2E 2042 6F78 2032 3139"
$0DA0 0099 A100 9A00 08FF DC00 0000 E600"
$00A0 0098 2800 CD00 1D22 2020 2020 2020"
$2020 2020 2020 2020 2020 2020 2020 2020"
$2020 2020 2020 2020 2047 6F6C 2990 1865"
$7461 2C20 4361 6C69 666F 726E 6961 2020"
$2039 3331 3136 0DA0 0099 A100 9A00 08FF"
$CC00 0000 E600 00A0 0098 2800 DD00 1D01"
$0DA0 0099 A100 9A00 08FF BC00 0000 E600"
$00A0 0098 2A10 2257 6520 616C 736F 206F"
$7065 7261 7465 2061 2062 756C 6C65 7469"
$6E20 626F 6172 6420 7329 D422 7973 7465"
$6D20 6174 2028 3830 3529 2036 3833 2D30"
$3332 322C 2062 6574 7765 656E 2074 29E6"
$0368 650D A000 99A1 009A 0008 FFAC 0000"
$00E6 0000 A000 9828 00FD 001D 2268 6F75"
$7273 206F 6620 363A 3030 2050 4D20 616E"
$6420 323A 3030 2041 4D2C 2050 6163 6929"
$D822 6669 6320 7469 6D65 2C20 7365 7665"
$6E20 6461 7973 2061 2077 6565 6B2E 2020"
$506C 6561 29D3 0373 650D A000 99A1 009A"
$0008 FF9C 0000 00E6 0000 A000 9828 010D"
$001D 1566 6565 6C20 6672 6565 2074 6F20"
$6361 6C6C 2069 6E21 A000 99A0 0097 A100"
$9600 0606 0000 0002 03A1 009A 0008 FFFD"
$0000 006D 0000 A000 980D 000A 2B73 1722"
$A920 3139 3838 2062 7920 436C 6966 666F"
$7264 2053 746F 7279 2061 6E64 2041 7474"
$6963 29AC 0920 536F 6674 7761 7265 A000"
$99A0 0097 A000 8DA0 0083 FF"

```

```

);
/*****
Window resource
*****/
resource 'WIND' (1001, purgeable, preload) (
    (0, 0, 200, 320),
    800,
    invisible,
    goaway,
    0x0,
    "untitled"
);
/*****
Scroll bar resource
*****/
resource 'CNTL' (1001, purgeable, preload) (
    (0, 0, 16, 50),
    1,
    invisible,
    1,
    1,
    scrollbarProc,
    0,
    ""
);
/*****
Bundle resources
*****/
resource 'BNDL' (1001, "Corners") (
    'CORN',
    0,
    ( /* array TypeArray: 2 elements */
        /* [1] */
        'ICN#',
        ( /* array IDArray: 1 elements */

```



```

/* [1] */
0, 1001,
),
/* [2] */
'FREF',
( /* array IDArray: 1 elements */
/* [1] */
0, 1001,
)
);

/*****
File reference resource
*****/
resource 'FREF' (1001, "CORN APPL", purgeable) (
'APPL',
0,
"
);

/*****
Icon list resource
*****/
resource 'ICON#' (1001, "APPL Icon", purgeable) (
/* array: 2 elements */
/* [1] */
$"0000 0000 7FFF FFFE 4040 0202 4040 0202"
$"4040 0202 47C0 03E2 4400 0022 4400 0022"
$"4400 0022 7C00 003E 4000 0002 4000 0002"
$"4000 0002 4000 0002 4000 0002 4000 0002"
$"4000 0002 4000 0002 4000 0002 4000 0002"
$"4000 0002 4000 0002 7C00 003E 4400 0022"
$"4400 0022 4400 0022 47C0 03E2 4040 0202"
$"4040 0202 4040 0202 7FFF FFFE",
/* [2] */
$"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
$"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
$"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
$"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
$"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
$"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
$"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
$"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
);

/*****
Icon resource
*****/
resource 'ICON' (257, purgeable) (
$"0000 0000 7FFF FFFE 4040 0202 4040 0202"
$"4040 0202 47C0 03E2 4400 0022 4400 0022"
$"4400 0022 7C00 003E 4000 0002 4000 0002"
$"4000 0002 4000 0002 4000 0002 4000 0002"
$"4000 0002 4000 0002 4000 0002 4000 0002"
$"4000 0002 4000 0002 7C00 003E 4400 0022"
$"4400 0022 4400 0022 47C0 03E2 4040 0202"
$"4040 0202 4040 0202 7FFF FFFE",
);

/*****
Finder resource
*****/
data 'CORN' (0, purgeable) (
$"50A9 2031 3938 382C 2062 7920 436C 6966"
/* P0 1988, by Clif */
$"666F 7264 2053 746F 7279 2026 2041 7474"
/* Ford Story & Att */
$"6963 2053 6F66 7477 6172 650D 502E 4F2E"
/* ic Software-P.O. */
$"2042 6F78 2032 3139 0D47 6F6C 6574 612C"
/* Box 219-Goleta, */
$"2043 616C 6966 6F72 6E69 6120 3933 3131"
);

```

```

/* California 9311 */
$"36"
/* 6 */
);

/*****
Multifinder resource
*****/
resource 'SIZE' (-1) (
saveScreen,
acceptSuspendResumeEvents,
enableOptionSwitch,
cannotBackground,
MultiFinderAware,
98304,
98304
);
/*****

```

Listing: Corners.make

```

*****/
# Corners.make
# (c) 1988, by Clifford Story & Attic Software
*****/
# compile resources
*****/
Corners ff Corners.make Corners.r
Rez Corners.r -append -o Corners

*****/
# compile common declarations
*****/
Common.p.o f Corners.make Common.p
Pascal Common.p

*****/
# compile and link WDEF
*****/
WDEF.p.o f Corners.make WDEF.p Common.p.o
Pascal WDEF.p
Corners ff Corners.make WDEF.p.o
Link -m WINDOWDEF -w -t 'APPL' -c 'CORN' d
-rt WDEF=50 -sn Main='Four Corners' d
WDEF.p.o d
"(Libraries)"Interface.o d
"(Libraries)"Runtime.o d
"(PLibraries)"PasLib.o d
"(PLibraries)"SANELib.o d
-o Corners

*****/
# compile ROM replacements
*****/
Patches.p.o f Corners.make Patches.p Common.p.o
Pascal Patches.p

*****/
# compile and link Corners
*****/
Corners.p.o f Corners.make Corners.p Patches.p.o
Pascal Corners.p
Corners ff Corners.make Corners.p.o
Link -w -t APPL -c 'CORN' d
Corners.p.o d
Patches.p.o d
"(Libraries)"Interface.o d
"(Libraries)"Runtime.o d
"(PLibraries)"PasLib.o d
"(PLibraries)"SANELib.o d
-o Corners
*****/

```



ABC's of Macintosh

Designer CDEFs



Kirk Chase
Assistant Editor
MacTutor
Volume 5, Number 4

Designer CDEFs

I was hunting around the office for software goodies a while back when I came across Prototyper™. I stuck the disk in to see what it was and was pleasantly surprised. Those hours spent getting the interface just right seemed over. Here was an environment where I could draw my windows, set up my menus and controls, and then let it generate the code for the interface.

Prototyper™ wasn't everything I wanted it to be, though. For example, it would generate code for things I didn't need. But it served my purpose of getting an interface/shell running quickly. (I hear version 2.0 is a lot better, but they haven't sent MacTutor an upgrade yet.) Anyway, I decided to test Prototyper™ out on graphic research question—Yahtzee (relaxation is important to mental health).

Well, after a bit, I had it up and running. I took it home and stuck it in my Mac for my recently married bride to look at. She took the mouse and started playing the game. Not as skilled in the mouse as I, whenever she went to click the checkbox beside each die to re-roll, she kept on missing or resetting it.

She asked me why she couldn't just click on the dice themselves instead of those little, tiny boxes. I was about to say that they weren't controls when I thought, "Why not? The user is always right." And so I went on thinking about the problem. CDEFs sounded like a lot of work, but they weren't for what I wanted. I just wanted a variation of the checkboxes. I decided just to alter the drawing routine of my checkboxes so that they just put a rounded rectangle around the die picture when selected. I found that this was quite easy to do and wrote up a small demonstration.

The Demo

The demo puts up five altered buttons (figure 1). The one in the middle is an altered checkbox made to look like a light switch. It is used to deactivate the controls on the left or right side. There are three radio buttons on the left which take on the appearance of a traffic light. The last button on the right is an altered button that cycles through the faces on a die (pictures taken from my Yahtzee application).

The CDEF

A CDEF is called through a function of the form:

```
function myCDEFProc(varCode : integer;  
                    theControl : ControlHandle;  
                    message : integer;  
                    param : longint) : longint;
```

It returns a longint telling what part of the control was pressed or zero if not pressed. The parameter *varCode* contains the variation code of the control. It lets you write one CDEF for a variety of controls. I chose though to use the *RefCon* field of

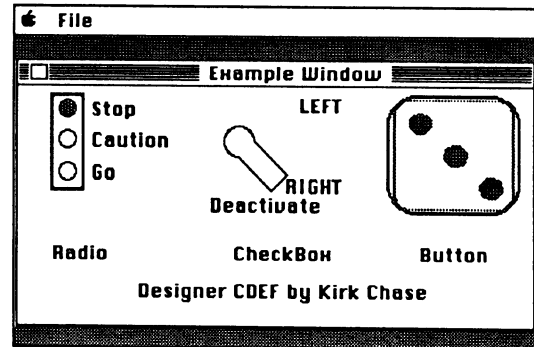


Figure 1. Designer CDEF

the control, *theControl*, to switch between routines. The *message* parameter tells the CDEF function what it needs to do. The last parameter, *param*, varies in meaning depending on *message*.

The unit that held my designer CDEF is called "myCDEF". A CDEF is supposed to handle a lot of tasks. It must take care of drawing the control according to its value and highlighting, as well as determining if the control was actually pressed. Therefore the main routine of our CDEF function is one where we check the *message* parameter that was passed, and do one of three routines: drawing, testing for a hit, and calculating the region. Other routines such as initializing or dragging the indicator are either done for us or not needed.

calcRgn

One message our altered CDEF needs to handle is calculating the region of the control. *Param* for this message is interpreted as a handle to a region. This is quite simple to do if your control is rectangular, you just make a call to *RectRgn* passing the region handle and the control's rectangle. To be honest, a rounded button is not really a strict rectangle, but the corners are generally too small to really matter.

testCntl

The next message we need to handle is whether or not the user pressed the mouse button while in our control. We first check to see if our control is active and then whether or not it was in the indicator (we have none). Finally we call *PinRect* on *param*, typecasted now to be a point, to see if we are actually in the control.

If we are in the control's rectangle, we switch on *ctrlRfCon* to see which type of control was pressed. Then we might want to do some further testing to see what part of the control was pressed, such as the page up or page down regions. Our CDEF has only one region so we send back the message *inButton*, if it is an altered button, or *inCheckbox*, if it was an altered checkbox or radio button.

drawCntl

I saved the largest routine for last. When we respond to a drawCntl message, we must draw the control with the appropriate value and highlighting. When we come in, we first check to see if the control is visible (no need to draw something that is invisible). Then we switch off the contrlRfCon field to handle each control type separately.

The Button

A normal button only shows it's state when pressed, i.e. highlighted. Then it goes back to normal. In my demo, I alter the button to not draw the title and draw a dark or light border around the control depending on whether it was highlighted or not. Then I dim the control if it is deactivated.

You may ask yourself, where are the faces of the die? They are not handled by the CDEF. I did this for two reasons. The first was to make a semi-generic control that put up a dark outline when pressed and a lighter one when not. The second reason was to show the effects of the picture when the control is deactivated.

The Checkbox

This simple drawing routine puts up the control's title and the On/Off title, in my case Left/Right. I might want to mention a nice routine, TextBox. If TextEdit is initialized, this routine saves the hassle of left/center/right justifying by passing it the title, the rectangle the title is to be drawn in, and the justification (teJustLeft, teJustCenter, teJustRight). Next, I see if the checkbox is on or off and draw the switch in that position with a thick pen or thin depending whether or not it is highlighted.

The Radio Controls

Although the radio controls are one type of control, each is drawn in a slightly different way. I first put up the title and then switch on contrlRfCon again to draw the traffic light box and set up the shade of the light. Then I draw the light according to the shade or override it with white if the control is not set. Then I dim the control if inactive.

You will notice that there is not a highlighted state, other than inactive. You do not need to put one in if you do not feel it is necessary. For that matter, there is no reason why you need to deactivate your control by dimming it. You could dim just part, dim it with a different pattern, or even draw something entirely different. Just remember not to go overboard and try to stick to the standard interface (I might have drawn a thicker circle). But if it seems natural to draw it in a nonstandard way, go ahead; that is after all the purpose to this article and all those little hooks.

Attaching Your Routine

The code to override the standard definition with yours is quite simple. Make sure your window is invisible to start off with so that you can attach your drawing routine before QuickDraw draws something. Then follow the example:

```
MyWindow := GetNewWindow(WindowID, NIL, Pointer(-1);
CDEFHandle := NewHandle(0); (Get temporary control handle)
```

```
CDEFHandle := Ptr(@myCDEFProc); (Point it to our routine)
CtrlHdl := GetNewControl(ButtonID, MyWindow);
HLock(Handle(CtrlHdl));
CtrlHdl^.ctrlDefProc := CDEFHandle; (Set to our proc)
HUnlock(Handle(CtrlHdl));
Show(MyWindow); (make it visible)
SelectWindow(MyWindow); (bring to front)
SetPort(MyWindow); (set the port to our window)
```

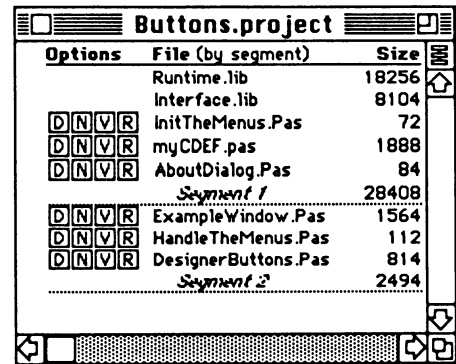
Further Notes

The die pictures are done by changing the min/max of the button to 0/5. When the button is pressed, the value is set accordingly, and the button's rectangle is invalidated. Then during the update event, the appropriate picture is retrieved according to the button's value and drawn. Then the controls are drawn, dimming the die face if inactive.

There are a couple notes of warning that you should be aware of. The first is that you must call DisposeWindow to dispose of your controls. Don't just quit, or you'll crash. The second has to do with debugging.

I worked this out in Lightspeed Pascal 2.0. Now I usually like to put a few writeln statements to the text window while developing. And I did so in my CDEF to trace it being called. Nothing appeared in my window. My tracing showed that I was indeed going where I should be with all the right values. What was wrong? Well it was those debugging statements. I figured that the writeln statements were setting the port to the text window and not resetting the port. Therefore my control drawing routines were being directed to the wrong port.

That's about it. Good luck!



Options	File (by segment)	Size
	Runtime.lib	18256
	Interface.lib	8104
☐ ☐ ☐ ☐ ☐	InitTheMenus.Pas	72
☐ ☐ ☐ ☐ ☐	myCDEF.pas	1888
☐ ☐ ☐ ☐ ☐	AboutDialog.Pas	84
	Segment 1	28408
☐ ☐ ☐ ☐ ☐	ExampleWindow.Pas	1564
☐ ☐ ☐ ☐ ☐	HandleTheMenus.Pas	112
☐ ☐ ☐ ☐ ☐	DesignerButtons.Pas	814
	Segment 2	2494

Figure 2. LSP Project

```
UNIT myCDEF;
```

INTERFACE

```
FUNCTION DesignerButtonProc (varCode : integer;
theControl : ControlHandle;
message : integer;
param : longint) : longint;
```

IMPLEMENTATION

```
FUNCTION DesignerButtonProc; ()
(Custom controls for our controls)
CONST
inactive = 255;
```

```

indicator = 254;
regbutton = 3;
regcheck = 4;
rStop = 5;
rCaution = 6;
rGo = 7;

PROCEDURE DodrawCntl (theButton : integer;
                      ControlHdl : ControlHandle;
                      thePart : longint);

VAR
  cRect, titleRect, designRect, tempRect, dimRect
  onBox, offBox : Rect;
  height, width, curve, start,
  titlewidth : integer;
  titlestr, tempstr : str255;

PROCEDURE DeactiveButton (theRect : rect);
(dim the button)
BEGIN
  PenPat(Gray);
  PenMode(PatBic);
  PaintRect(theRect);
  PenNormal;
END;

BEGIN (of main drawing routines)
IF ControlHdl^.ctrlVis <> 0 THEN (visible?)
  BEGIN
    HLock(Handle(ControlHdl));
    cRect := ControlHdl^.ctrlRect;
    (get the rectangle of control)
    CASE ControlHdl^.ctrlRfCon OF
      (switch on button type)
      regular button)
      BEGIN
        PenSize(2, 2);
        curve := (cRect.bottom - cRect.top) DIV 2;
        IF ControlHdl^.ctrlHilite > 0 THEN
          (hlight button?)
          PenPat(black)
        ELSE
          PenPat(dkgray);
          FrameRoundRect(cRect, curve, curve);
          (draw button)
          PenNormal;
          IF ControlHdl^.ctrlHilite = inactive
          THEN (inactive?)
          BEGIN
            tempRect := cRect;
            InsetRect(tempRect, 2, 2);
            DeactiveButton(tempRect);
          END;
        END;

      regcheck : (checkbox)
      BEGIN
        EraseRect(cRect);
        titleRect := cRect;
        titleRect.top := titleRect.bottom - 12;
        designRect := cRect;
        designRect.bottom := designRect.bottom - 12;
        width := titleRect.right -
          titleRect.left;
        height := designRect.bottom -
          designRect.top;

        (draw control title)
        TextFont(systemFont);
        titlewidth := StringWidth(ControlHdl^.
          ctrlTitle);
        titlestr := ControlHdl^.ctrlTitle;
        TextBox(Pointer(ord(@titlestr) + 1),

```

```

          length(titlestr), titleRect,
          teJustLeft);

        (draw ON/OFF, or LEFT/RIGHT, titles)
        tempRect := designRect;
        tempRect.left := tempRect.left + 20;
        tempRect.bottom := tempRect.top + 12;
        tempstr := 'LEFT';
        TextBox(Pointer(ord(@tempstr) + 1),
          length(tempstr), tempRect,
          teJustRight);
        tempRect.bottom := designRect.bottom;
        tempRect.top := tempRect.bottom - 12;
        tempstr := 'RIGHT';
        TextBox(Pointer(ord(@tempstr) + 1),
          length(tempstr), tempRect,
          teJustRight);
        TextFont(applFont);

        (start drawing switch)
        tempRect := designRect;
        tempRect.right := tempRect.left + 40;
        WITH tempRect DO (draw Switch)
        BEGIN
          width := right - left;
          height := bottom - top;
          onBox.top := top + (height DIV 2) - 10;
          onBox.bottom := onBox.top + 20;
          onBox.left := left + (width DIV 2) - 10;
          onBox.right := onBox.left + 20;

          IF ControlHdl^.ctrlHilite > 0
          THEN (Hilite?)
            PenSize(2, 2)
          ELSE
            PenSize(1, 1);
          IF ControlHdl^.ctrlValue > 0
          THEN
            BEGIN (draw on)
              FrameArc(onBox, 0, -270);
              MoveTo(left + (width DIV 2),
                onBox.top);
              Line(20, -20);
              Line(10, 10);
              LineTo(onBox.right, top +
                (height DIV 2));
            END
          ELSE
            BEGIN (draw off)
              FrameArc(onBox, 90, -270);
              MoveTo(onBox.right, top +
                (height DIV 2));
              Line(20, 20);
              Line(-10, 10);
              LineTo(left + (width DIV 2),
                onBox.bottom);
            END;
          END; (of with)
        END;

      rStop, rCaution, rGo : (radio button)
      BEGIN
        designRect := cRect;
        titleRect := designRect;
        titleRect.left := titleRect.left + 25;
        titleRect.top := titleRect.top + 3;
        designRect.right := designRect.left + 20;

        (draw title of control)
        TextFont(systemFont);
        titlestr := ControlHdl^.ctrlTitle;
        TextBox(Pointer(ord(@titlestr) + 1),
          length(titlestr), titleRect,

```

```

teJustLeft);
TextFont(app1Font);
height := designRect.bottom -
designRect.top;

(switch on RefCon for radio button: stop, caution, and go...
this is for radio button specific stuff)

```

```

CASE ControlHdl^^.ctrlRfCon OF
  rStop :
    BEGIN
      (draw box)
      PenSize(2, 2);
      MoveTo(designRect.left,
designRect.bottom);
      LineTo(designRect.left,
designRect.top);
      LineTo(designRect.right,
designRect.top);
      LineTo(designRect.right,
designRect.bottom);
      PenNormal;
      PenPat(dkgray); (light color)
    END;
  rCaution :
    BEGIN
      (draw box)
      PenSize(2, 2);
      MoveTo(designRect.left,
designRect.bottom);
      LineTo(designRect.left,
designRect.top);
      MoveTo(designRect.right,
designRect.top);
      LineTo(designRect.right,
designRect.bottom);
      PenNormal;
      PenPat(ltgray); (light color)
    END;
  rGo :
    BEGIN
      (draw box)
      PenSize(2, 2);
      MoveTo(designRect.left,
designRect.top);
      LineTo(designRect.left,
designRect.bottom);
      LineTo(designRect.right,
designRect.bottom);
      LineTo(designRect.right,
designRect.top);
      PenNormal;
      PenPat(gray); (light color)
    END;
  OTHERWISE
    ;
END; ( of case )

(draw light)
onBox := designRect;
InsetRect(onBox, 4, 4);
onBox.left := onBox.left + 1;
onBox.right := onBox.right + 1;
IF ControlHdl^^.ctrlValue = 0 THEN
  (override pattern if off)
  PenPat(white);
  PaintOval(onBox);
  PenNormal;
  FrameOval(onBox);

  IF ControlHdl^^.ctrlHilite = inactive
  THEN (deactivate control)

```

```

    BEGIN
      tempRect := cRect;
      DeactivateButton(tempRect);
    END;
  END;
END; (of button)

HUnlock(Handle(ControlHdl));
END; (of a visible control)

FUNCTION DotestCnt1 (theButton : integer;
ControlHdl : ControlHandle;
thePoint : point) : longint;
(Test what part of control)
BEGIN
  DotestCnt1 := 0; (initial value)
  IF ControlHdl^^.ctrlHilite <> inactive THEN
    (active control?)
    BEGIN
      IF ControlHdl^^.ctrlHilite <> indicator
      THEN (indicator?)
        BEGIN
          IF PtInRect(thePoint, ControlHdl^^
.ctrlRect) THEN (in control's rect)
            BEGIN
              CASE ControlHdl^^.ctrlRfCon OF
                (switch on control type... here is where
                you send a code back as to what type
                of control and what part of that
                control was pressed. We only have
                controls with one part, so no further
                testing is needed)
                regbutton :
                  DotestCnt1 := inButton;
                regcheck, rStop, rCaution, rGo :
                  DotestCnt1 := inCheckBox;
                OTHERWISE
                  ;
                END;
              END;
            END
          ELSE
            DotestCnt1 := indicator;
          END;
        END;
      END;
    END;
  END;

PROCEDURE DocalcCRgns (theButton : integer;
ControlHdl : ControlHandle;
RegionHdl : RgnHandle);
(calculate region of control)
BEGIN
  (easy as pie if rectangular)
  RectRgn(RegionHdl, ControlHdl^^.ctrlRect);
END;

BEGIN (main proc routine)
  DesignerButtonProc := 0; (initialize the result)

  (switch on what we are to do)
  CASE message OF
    drawCnt1 :
      DodrawCnt1(varCode, theControl, param); (draw)
    testCnt1 :
      DesignerButtonProc := DotestCnt1(varCode,
theControl, Point(param)); (test)
    calcCRgns :
      DocalcCRgns(varCode, theControl,
RgnHandle(param)); (region)
    OTHERWISE
      ;
  END; (of case)
END;

```

END.

```
PROGRAM DesignerButtons;
(Program name: DesignerButtons.Pas )
(Function: This is the main module for this program. )
(History: 12/19/88 Original by Prototyper. )
( Modified 1/6/89 )
```

USES

```
AboutDialog, ExampleWindow, InitTheMenus,
HandleTheMenus;
```

CONST

```
sleep = 10;
mBarHeightGlobal = $BAA;
sBarWidth = 16;
SuspendEvt = 15;
```

VAR

```
(Main variables)
myEvent: EventRecord; (Event record for all events)
doneFlag: boolean; (Exit program flag)
code: integer; (Determine event type)
whichWindow, MainWindow: WindowPtr;
mResult: longint;
theMenu, theItem: integer;
chCode: integer; (Key code)
ch: char; (Key pressed in Ascii)
doIt: Boolean; (Event Loop boolean)
tempPort: GrafPtr;
DragArea, screen: rect;
mBarHeight: integer;
MemoryPtr: ^Integer;
HiByte: byte;
bit0: LongInt;
sysresult: boolean;
ResumePeek: WindowPeek;
SuspendPeek: WindowPeek;
```

PROCEDURE crash;

```
BEGIN
ExitToShell;
END;
```

BEGIN

```
(main )
MoreMasters;
InitGraf(@thePort); (Quickdraw Init)
InitFonts; (Font manager init)
InitWindows; (Window manager init)
InitMenus; (Menu manager init)
TEInit; (Text edit init)
InitDialogs(@crash); (Dialog manager)
FlushEvents(everyEvent, 0);
InitCursor; (Make an arrow cursor)
doneFlag := FALSE; (Do not exit program yet)
```

```
MemoryPtr := pointer(mBarHeightGlobal);
mBarHeight := MemoryPtr^;
screen := ScreenBits.Bounds; (current screen device)
SetRect(DragArea, Screen.left + 4, Screen.top +
mBarHeight + 4, Screen.right - 4, Screen.bottom - 4);
```

```
Init_My_Menus; (Initialize menu bar)
Init_ExampleWindow; (Initialize the window )
Open_ExampleWindow(MainWindow); (Open the window )
```

REPEAT

```
(main event loop)
DoIt := WaitNextEvent(EveryEvent, myEvent, sleep,
NIL); (no mouse tracking)
```

IF DoIt THEN

BEGIN

```
code := FindWindow(myEvent.where, whichWindow); (Get
```

which window the event happened in)

CASE myEvent.what OF

MouseDown: (Mouse button pressed)

BEGIN

IF (code = inMenuBar) THEN

BEGIN

```
(Get menu selection )
mResult := MenuSelect(myEvent.Where);
theMenu := HiWord(mResult); ( menu)
theItem := LoWord(mResult); (item)
Handle_My_Menu(doneFlag, theMenu,
theItem);
```

END;

(End of inMenuBar)

IF (code = inContent) THEN

BEGIN

```
(Handle hit inside a window)
IF (whichWindow <> FrontWindow) THEN
SelectWindow(whichWindow)
```

ELSE

BEGIN

```
(Handle the button)
SetPort(whichWindow);
Do_ExampleWindow(myEvent);
```

END;

END;

(End of inContent)

IF (code = inSysWindow) THEN

(See if a DA selection)

```
SystemClick(myEvent, whichWindow);
```

IF (code = inDrag) THEN

```
DragWindow(whichWindow, myEvent.Where,
DragArea);
```

IF (code = inGoAway) THEN

```
IF TrackGoAway(whichWindow,
myEvent.Where) THEN
```

BEGIN

```
HideWindow(whichWindow);
doneFlag := true;
```

END;

END;

(End of MouseDown)

KeyDown, AutoKey: (Handle key inputs)

BEGIN

WITH myevent DO

BEGIN

```
( )
chCode := BitAnd(message,
CharCodeMask); (Get character)
ch := CHR(chCode);
(Change to ASCII)
```

IF (Odd(modifiers DIV CmdKey)) THEN

BEGIN

```
( )
mResult := MenuKey(ch);
theMenu := HiWord(mResult);
theItem := LoWord(mResult);
IF (theMenu <> 0) THEN
(See if a list was selected)
Handle_My_Menu(doneFlag,
theMenu, theItem);
```

END;

END;

(End for with)

END;

(End for KeyDown, AutoKey)

UpDateEvt: (Update event for a window)

BEGIN

```
(Handle the update)
GetPort(tempPort);
whichWindow := WindowPtr(myEvent.message);
(Get the window the update is for)
SetPort(whichWindow);
BeginUpdate(whichWindow);
(Set the clipping to the update area)
Update_ExampleWindow(whichWindow);
(Update this window)
```

```

    EndUpdate(whichWindow);
    (Return to normal clipping area)
    SetPort(tempPort);
END; (End of UpDateEvt)

ActivateEvt: (Window activated event)
BEGIN (Handle the activation)
    whichWindow := WindowPtr(myEvent.message);
    (Get the window to be activated)
    IF odd(myEvent.modifiers) THEN
        (Activate and not DeActivate)
        SelectWindow(whichWindow);
        (Activate the window by selecting it)
    END; (End of ActivateEvt)

SuspendEvt:
BEGIN
    bit0 := 31; (convert 68000 to toolbox)
    (check for resume event)
    IF Odd(myEvent.message) THEN
        BEGIN (resume)
            whichWindow := FrontWindow;
            IF whichWindow = MainWindow THEN
                BEGIN
                    SetPort(whichWindow);
                    InvalRect(whichWindow^.portRect);
                    SelectWindow(whichWindow);
                END;
            IF FrontWindow <> NIL THEN
                BEGIN (DA check)
                    ResumePeek :=
                        WindowPeek(FrontWindow);
                    IF ResumePeek^.windowKind < 0
                        THEN (DA)
                            BEGIN (da)
                                myEvent.what := activateEvt;
                                BitSet(@myEvent.modifiers,
                                    bit0);
                                sysresult :=
                                    SystemEvent(myEvent);
                                END; (da)
                            END; (DA check)
                        (end of activate Event)
                    END (of resume)
                ELSE
                    BEGIN (suspend)
                        (de-activate Event)
                        whichWindow := FrontWindow;
                        IF whichWindow = MainWindow THEN
                            BEGIN
                                SetPort(whichWindow);
                                InvalRect(whichWindow^.portRect);
                            END;
                        IF FrontWindow <> NIL THEN
                            BEGIN (DA check)
                                SuspendPeek :=
                                    WindowPeek(FrontWindow);
                                IF SuspendPeek^.windowKind < 0
                                    THEN
                                        BEGIN (da)
                                            myEvent.what := activateEvt;
                                            BitClr(@myEvent.modifiers,
                                                bit0);
                                            sysresult :=
                                                SystemEvent(myEvent);
                                            END; (da)
                                        END; (DA check)
                                    END; (suspend)
                                END;
                            OTHERWISE
                                BEGIN
                                    (End of otherwise)
                                END;
                            END;
                        OTHERWISE
                            BEGIN
                                (End of otherwise)
                            END;
                        END;
                    END;
                END;
            END;
        END;
    END;
END;

```

```

    END; (End of case)
    END; (end of waitNextEvent)
    UNTIL doneFlag; (End of the event loop)
    DisposeWindow(MainWindow);
END. (End of the program)

```

UNIT InitTheMenus;

INTERFACE

PROCEDURE Init_My_Menus;

```

VAR
    AppleMenu: MenuHandle;
    FileMenu: MenuHandle;

```

IMPLEMENTATION

PROCEDURE Init_My_Menus; (Initialize the menus)

```

CONST
    Menu1 = 201; (Apple Menu resource ID)
    Menu2 = 202; (File Menu resource ID)
BEGIN (Start of Init_My_Menus)
    ClearMenuBar; (Clear any old menu bars)
    AppleMenu := GetMenu(Menu1); (Get apple menu)
    AddResMenu(AppleMenu, 'DRVR'); (Add in DAs)
    InsertMenu(AppleMenu, 0);
    FileMenu := GetMenu(Menu2); (File menu)
    InsertMenu(FileMenu, 0);
    DrawMenuBar; (Draw the menu bar)
END; (End of procedure Init_My_Menus)
END. (End of Unit)

```

UNIT AboutDialog;

```

(File name:AboutDialog.Pas )
(Function: Handle a dialog)

```

INTERFACE

PROCEDURE D_AboutDialog;

IMPLEMENTATION

```

CONST (item numbers for controls in Dialog)
    I_OK = 1;
VAR
    ExitDialog: boolean; (Flag used to exit Dialog)

```

PROCEDURE D_AboutDialog;

```

VAR
    GetSelection: DialogPtr; (Name of dialog)
    itemHit: Integer; (Get selection)
BEGIN (Start of dialog handler)
    GetSelection := GetNewDialog(2, NIL, Pointer(-1));
    (Bring in the dialog resource)
    ShowWindow(GetSelection); (Open a dialog box)
    SelectWindow(GetSelection); (Lets see it)
    SetPort(GetSelection);

    ExitDialog := FALSE; (Do not exit loop yet)
    REPEAT (Start of dialog handle loop)
        ModalDialog(NIL, itemHit);
        IF (itemHit = I_OK) THEN (Handle Button)
            BEGIN
                ExitDialog := TRUE; (Exit the dialog)
            END; (End for this item selected)
        UNTIL ExitDialog; (Handle until exit selected)
        DisposDialog(GetSelection);
    END; (End of procedure)
END. (End of unit)

```

UNIT ExampleWindow;

(File name: ExampleWindow.Pas)
(Function: Handle a Window)

INTERFACE

USES

myCDEF;

PROCEDURE Init_ExampleWindow;

PROCEDURE Open_ExampleWindow (VAR theWindow: WindowPtr);

PROCEDURE Update_ExampleWindow (whichWindow: WindowPtr);

PROCEDURE Do_ExampleWindow (myEvent: EventRecord);

IMPLEMENTATION

CONST

I_Buttonx1 = 3; (Button ID)
I_Checkboxx2 = 4; (Checkbox ID)
I_Radio2 = 6; (Radio ID)
I_Radio3 = 7; (Radio ID)
I_Radio1 = 5; (Radio ID)

VAR

MyWindow: WindowPtr; (Window pointer)
tempRect: Rect; (Temporary rectangle)
CtrlHandle, buttonHdl: controlhandle;
sTemp: Str255; (Get text entered, temp holding)
R1Control: ARRAY[1..3] OF ControlHandle;
(Radio button handles for group 1)

PROCEDURE Init_ExampleWindow;

BEGIN

MyWindow := NIL; (we are not valid yet)

END;

(Update our window, someone uncovered a part of us)

PROCEDURE UpDate_ExampleWindow;

CONST

picBaseID = 144;

VAR

SavePort: WindowPtr; (Place to save the last port)
pictRect: rect;
Pic_Handle: PicHandle;
pictindex: integer;

BEGIN (Start of Window update routine)

IF (MyWindow <> NIL) AND (MyWindow = whichWindow)

THEN

BEGIN

TextFont(systemFont);

(Draw a string of text, Static Text)

SetRect(tempRect, 266, 100, 312, 116);

sTemp := 'Button';

TextBox(Pointer(ord(sTemp) + 1), length(sTemp),
tempRect, teJustLeft);

(Draw a string of text, Static Text)

SetRect(tempRect, 151, 100, 217, 116);

sTemp := 'CheckBox';

TextBox(Pointer(ord(sTemp) + 1), length(sTemp),
tempRect, teJustLeft);

(Draw a string of text, Static Text)

SetRect(tempRect, 36, 100, 77, 116);

sTemp := 'Radio';

TextBox(Pointer(ord(sTemp) + 1), length(sTemp),
tempRect, teJustLeft);

(Draw a string of text, Static Text)

SetRect(tempRect, 76, 125, 287, 141);

sTemp := 'Designer CDEF by Kirk Chase';

TextBox(Pointer(ord(sTemp) + 1), length(sTemp),
tempRect, teJustLeft);

TextFont(app1Font);
pictRect := buttonHdl^.controlRect;
InsetRect(pictRect, 2, 2);
pictindex := GetCtlValue(buttonHdl);
Pic_Handle := GetPicture(picBaseID + pictindex);
(Get Picture into memory)

IF (Pic_Handle <> NIL) **THEN**

DrawPicture(Pic_Handle, pictRect);

DrawControls(MyWindow); (Draw controls)

END; (End for if (MyWindow<>nil))

END; (End of procedure)

(Open our window and draw everything)

PROCEDURE Open_ExampleWindow;

VAR

CDEFHandle: handle;

chkhdl: controlhandle;

BEGIN (Start of Window open routine)

IF (MyWindow = NIL) **THEN**

BEGIN

MyWindow := GetNewWindow(1, NIL, Pointer(-1));

(Get the window from the resource file)

theWindow := MyWindow;

(get a handle to our CDEF Proc)

CDEFHandle := NewHandle(0);

CDEFHandle^ := Ptr(@DesignerButtonProc);

(Make a button, Button)

CtrlHandle := GetNewControl(I_Buttonx1,
MyWindow);

HLock(Handle(CtrlHandle));

CtrlHandle^.controlDefProc := CDEFHandle;

HUnlock(Handle(CtrlHandle));

CtrlHandle^.controlHilite := 255;

buttonHdl := CtrlHandle;

(Make a checkbox, Checkbox)

CtrlHandle := GetNewControl(I_Checkboxx2,
MyWindow); (Make a new checkbox)

HLock(Handle(CtrlHandle));

CtrlHandle^.controlDefProc := CDEFHandle;

HUnlock(Handle(CtrlHandle));

chkhdl := CtrlHandle;

(Make a radio button, Radio1)

R1Control[1] := GetNewControl(I_Radio1,
MyWindow);

HLock(Handle(CtrlHandle));

R1Control[1]^..controlDefProc := CDEFHandle;

HUnlock(Handle(CtrlHandle));

(Make a radio button, Radio2)

R1Control[2] := GetNewControl(I_Radio2,
MyWindow);

HLock(Handle(CtrlHandle));

R1Control[2]^..controlDefProc := CDEFHandle;

HUnlock(Handle(CtrlHandle));

(Make a radio button, Radio3)

R1Control[3] := GetNewControl(I_Radio3,
MyWindow);

HLock(Handle(CtrlHandle));

R1Control[3]^..controlDefProc := CDEFHandle;

HUnlock(Handle(CtrlHandle));

ShowWindow(MyWindow);

SelectWindow(MyWindow); (window to front)

SetPort(MyWindow); (write into our window)

UpDate_ExampleWindow(MyWindow);


```

END      (End for if (MyWindow<>nil))
ELSE
    SelectWindow(MyWindow);

END;      (End of procedure)

(Handle action to our window, like controls)
PROCEDURE Do_ExampleWindow;
CONST
    HiliteValue = 10;
    inactive = 255;
    active = 0;
    left = 0;
    right = 1;

VAR
    RefCon: integer; (RefCon for controls)
    code: integer; (Location of event)
    theValue: integer; (Current value of a control)
    whichWindow: WindowPtr; (window for event)
    myPt, myPtGlobal: Point;
    theControl: ControlHandle;
    badRect: rect;

PROCEDURE Do_A_Button;

BEGIN
    HiliteControl(theControl, HiliteValue);
    (Darken the button)
    RefCon := GetCRefCon(theControl);
    (get control refcon)

CASE RefCon OF (Select correct button)
    I_Buttonx1: (Button, button)
        BEGIN (start for this button)
            theValue := GetCtlValue(theControl);
            theValue := (theValue + 1) MOD 6;
            SetCtlValue(theControl, theValue);
            (Set button to new value)
            badRect := buttonhd1^.ctrlRect;
            InvalRect(badRect);
        END; (end for this button)

        OTHERWISE (allow other buttons, trap for debug)
        ;

    END; (end of case)

    HiliteControl(theControl, active);
END; (Handle a button being pressed)

PROCEDURE Do_A_Checkbox;
(Handle a checkbox being pressed)
VAR
    Index: integer; (Index used for radios)

PROCEDURE ClearRadioGroup;
(Routine to clear radios in group 1)
VAR
    Index: integer; (Index used for radios)
BEGIN (Start of the clear routine)
    FOR Index := 1 TO 3 DO (Step thru all radios)
        SetCtlValue(R1Control[Index], 0);
        (Set this radio to zero)
    END; (End of the clear routine)

BEGIN (Handle a checkbox being pressed)
    RefCon := GetCRefCon(theControl);
    theValue := GetCtlValue(theControl);
    theValue := (theValue + 1) MOD 2; (Change value)

CASE RefCon OF (Select correct button)
    I_Checkboxx2: (Checkbox, checkbox)

```

```

BEGIN (start for this button)
    SetCtlValue(theControl, theValue);
    (Set checkbox to new value)
    badRect := buttonhd1^.ctrlRect;
    InvalRect(badRect);
CASE theValue OF (left or right controls)
    left:
        BEGIN
            HiliteControl(buttonHd1, inactive);
            HiliteControl(R1Control[1], active);
            HiliteControl(R1Control[2], active);
            HiliteControl(R1Control[3], active);
        END;
    right:
        BEGIN
            HiliteControl(buttonHd1, active);
            HiliteControl(R1Control[1], inactive);
            HiliteControl(R1Control[2], inactive);
            HiliteControl(R1Control[3], inactive);
        END;
    OTHERWISE
        ;
END;
END; (end for this checkbox)

I_Radio1, I_Radio2, I_Radio3:
(radio buttons)
BEGIN (start for this radio button)
    ClearRadioGroup;
    SetCtlValue(theControl, 1);
END; (end for this radio button)
OTHERWISE
;
END; (end of case)
END; (Handle a checkbox being pressed)

BEGIN (Start of Window handler)
IF (MyWindow <> NIL) THEN
    BEGIN
        code := FindWindow(myEvent.where, whichWindow);
        IF (myEvent.what =MouseDown) AND (MyWindow =
            whichWindow) THEN
            BEGIN
                myPt := myEvent.where; (Get mouse position)
                myPtGlobal := myPt;
                GlobalToLocal(myPt);
            END;
            IF (MyWindow = whichWindow) AND (code =
                inContent) THEN (for our window)
                BEGIN
                    code := FindControl(myPt, whichWindow,
                        theControl); (Get type of control)
                    IF (code <> 0) THEN (Check type)
                        code := TrackControl(theControl, myPt,
                            NIL); (Track the control)
                    IF code = inButton THEN
                        Do_A_Button; (Do buttons)
                    IF code = inCheckBox THEN
                        Do_A_Checkbox; (Do checkboxes)
                    END; (End for if (MyWindow=whichWindow))
                END; (End for if (MyWindow<>nil))
            END; (End of procedure)
        END. (End of unit)

UNIT HandleTheMenus;

INTERFACE

USES
    AboutDialog, ExampleWindow, InitTheMenus;

PROCEDURE Handle_My_Menu (VAR doneFlag: boolean;

```

```
theMenu, theItem: integer); (Handle menu selection)
```

IMPLEMENTATION

```
PROCEDURE Handle_My_Menu;
```

```
CONST
```

```
  L_Apple = 201; (Menu list)  
  C_About = 1;  
  L_File = 202; (Menu list)  
  C_Quit = 1;
```

```
VAR
```

```
  DNA: integer; (For opening DAs)  
  BoolHolder: boolean; (For SystemEdit result)  
  DAName: Str255; (For getting DA name)  
  SavePort: GrafPtr; (Save current port)
```

```
BEGIN
```

```
  CASE theMenu OF (Do selected menu list)
```

```
    L_Apple:
```

```
      BEGIN
```

```
        IF theItem = C_About THEN
```

```
          D_AboutDialog (Call a dialog)
```

```
        ELSE (Handle the DAs)
```

```
          BEGIN
```

```
            GetPort(SavePort); (Save port)  
            GetItem(AppleMenu, theItem, DAName);  
            DNA := OpenDeskAcc(DAName); (Open DA )  
            SetPort(SavePort); (Restore port)
```

```
          END;
```

```
        END; (of L_Apple)
```

```
    L_File:
```

```
      BEGIN
```

```
        IF theItem = C_Quit THEN
```

```
          doneFlag := true;
```

```
      END;
```

```
    OTHERWISE
```

```
      BEGIN
```

```
      END;
```

```
  END;
```

```
  HiliteMenu(0); (Turn menu selection off)
```

```
END; (End of procedure Handle_My_Menu)
```

```
END. (End of unit)
```

```
*****
```

```
*  
* RMaker resource file sources.  
* File: DesignerButtons.R  
* History: 12/19/88 Original by Prototyper.  
*
```

```
DesignerButtons.RSRC
```

```
???KCBx
```

```
Type KCBx = STR
```

```
ID,0
```

```
CDEF Demo \0D0 by Kirk Chase \0Dver 19 DEC 1989
```

```
* Multifinder Menu for Quit Cmd
```

```
Type mstr = STR
```

```
File,100
```

```
File
```

```
* Multifinder Quit name
```

```
Type mstr = STR
```

```
Quit,101
```

```
Quit
```

```
Type FREF
```

```
CDEF,128
```

```
APPL 0
```

```
Type BNDL
```

```
Bundle,128
```

```
KCBx 0
```

```
ICN#
```

```
0 128
```

```
FREF
```

```
0 128
```

```
Type SIZE = GNRL
```

```
MultiFinder,-1
```

```
.H
```

```
48000 ;; $4800 = bits 14 & 11 set
```

```
.L
```

```
80000 ;; (recommended)
```

```
.L
```

```
50000 ;; (minimum)
```

```
.I
```

```
Type vers = GNRL
```

```
, 1
```

```
.H
```

```
01;; byte vers # in BCD
```

```
10;; byte vers part 2 & 3
```

```
50;; byte release stage $50=release
```

```
00;; byte stage of non-release
```

```
00 00 ;; integer country 0=US
```

```
.P
```

```
V1.1 (US)
```

```
.P
```

```
CDEF Demo 1.1, 0 by Kirk Chase for MacTutor 1989
```

```
Type vers = GNRL
```

```
, 2
```

```
.H
```

```
05;; byte vers # in BCD
```

```
30;; byte vers part 2 & 3
```

```
50;; byte release stage $50=release
```

```
00;; byte stage of non-release
```

```
00 00 ;; integer country 0=US
```

```
.P
```

```
V5.3
```

```
.P
```

```
MacTutor Volume 5 Number 3
```

```
* This is the definition for the DIALOG.
```

```
*
```

```
Type DLOG
```

```
About ,2
```

```
;;Resource ID
```

```
About Dialog
```

```
;;Dialog title
```

```
186 87 313 411
```

```
;;Top Left Bottom Right
```

```
Visible NoGoAway
```

```
;;Visible GoAway
```

```
1
```

```
;;ProcID, dialog def ID
```

```
2
```

```
;;Refcon, reference value
```

```
2
```

```
;;ID of item list
```

```
* This is the DIALOG or ALERT item list.
```

```
*
```

```
Type DITL
```

```
About Items ,2
```

```
;;Resource ID
```

```
3
```

```
;;Number of controls in list
```

```
Button Enabled
```

```
;;Push button
```

```
75 181 111 247
```

```
;;Top Left Bottom Right
```

```
OK
```

```
;;message
```

```
StaticText
```

```
;;Static text
```

```
40 11 56 297
```

```
;;Top Left Bottom Right
```

```
Portions of code developed by Prototyper
```

```
StaticText
```

```
;;Static text
```

```
15 6 36 307    ;;Top Left Bottom Right
Designer CDEF by Kirk Chase For MacTutor\00
```

```
* This is the definition for the MENU.
```

```
*
Type MENU
Apple      ,201    ;;Resource ID
\14        ;;APPLE menu title
About Designer CDEF ...
(-
```

```
File      ,202    ;;Resource ID
File      ;;menu title
Quit/Q    ;;item title
```

```
* This is the definition for the WINDOW.
```

```
*
Type WIND
Kirk Chase ,1    ;;Resource ID
Example Window ;;Window title
55 17 210 373   ;;Top Left Bottom Right
InVisible GoAway ;;Visible GoAway
4              ;;ProcID, Window def ID
1              ;;Refcon, reference value
```

```
* This is the CONTROL item list.
```

```
*
Type CNTL
Button ,3    ;;Resource ID
Button    ;;Title for a Button
5 246 81 332 ;;Top Left Bottom Right
Visible   ;;Initially visible
0         ;;ProcID (Control definition ID)
3         ;;RefCon (reference value)
0 5 0     ;;Min Max Value
```

```
Checkbox ,4    ;;Resource ID
Deactivate ;;Title for a Checkbox
5 136 80 221  ;;Top Left Bottom Right
Visible     ;;Initially visible
1           ;;ProcID (Control definition ID)
4           ;;RefCon (reference value)
0 1 0      ;;Min Max Value
```

```
Radio Caution ,6 ;;Resource ID
Caution
25 21 45 106
Visible
2
6
0 1 0
```

```
Radio Go ,7    ;;Resource ID
Go
45 21 65 106
Visible
2
7
0 1 0
```

```
Radio Stop ,5 ;;Resource ID
Stop
5 21 25 106
Visible
2
5
0 1 1
```

```
* PICTs for button
Type PICT = GNRL
,149
.H
008B 0007 0007 0068 0068 1101 A000 82A0
```

```
008C 0100 0A00 0700 0700 6800 680A 0000
0000 0000 0000 0B00 1B00 1B44 0009 0009
0066 0066 0700 0200 0248 0AFF FFFF FFFF
FFFF FF54 0015 0015 0027 0027 0700 0100
0158 5400 2F00 1400 4200 2758 5400 1400
4A00 2700 5D58 5400 2F00 4A00 4200 5D58
5400 4A00 4A00 5D00 5D58 5400 4A00 1400
5D00 2758 A000 8DA0 0083 FF
```

```
Type PICT = GNRL
,148
.H
0081 0007 0007 0068 0068 1101 A000 82A0
008C 0100 0A00 0700 0700 6800 680A 0000
0000 0000 0000 0B00 1B00 1B44 0009 0009
0066 0066 0700 0200 0248 0AFF FFFF FFFF
FFFF FF54 004B 0015 005D 0027 0700 0100
0158 5400 4A00 4A00 5D00 5D58 5400 1400
1400 2700 2758 5400 1400 4A00 2700 5D58
5400 2F00 2F00 4200 4258 A000 8DA0 0083
FF
```

```
Type PICT = GNRL
,147
.H
0077 0007 0007 0068 0068 1101 A000 82A0
008C 0100 0A00 0700 0700 6800 680A 0000
0000 0000 0000 0B00 1B00 1B44 0009 0009
0066 0066 0700 0200 0248 0AFF FFFF FFFF
FFFF FF54 0015 0015 0027 0027 0700 0100
0158 5400 1400 4A00 2700 5D58 5400 4A00
1400 5D00 2758 5400 4A00 4A00 5D00 5D58
A000 8DA0 0083 FF
```

```
Type PICT = GNRL
,146
.H
006D 0007 0007 0068 0068 1101 A000 82A0
008C 0100 0A00 0700 0700 6800 680A 0000
0000 0000 0000 0B00 1B00 1B44 0009 0009
0066 0066 0700 0200 0248 0AFF FFFF FFFF
FFFF FF54 0015 0015 0027 0027 0700 0100
0158 5400 2F00 2F00 4200 4258 5400 4A00
4A00 5D00 5D58 A000 8DA0 0083 FF
```

```
Type PICT = GNRL
,145
.H
0063 0007 0007 0068 0068 1101 A000 82A0
008C 0100 0A00 0700 0700 6800 680A 0000
0000 0000 0000 0B00 1B00 1B44 0009 0009
0066 0066 0700 0200 0248 0AFF FFFF FFFF
FFFF FF54 004B 004B 005D 005D 0700 0100
0158 5400 1400 1400 2700 2758 A000 8DA0
0083 FF
```

```
Type PICT = GNRL
,144
.H
0059 0007 0007 0068 0068 1101 A000 82A0
008C 0100 0A00 0700 0700 6800 680A 0000
0000 0000 0000 0B00 1B00 1B44 0009 0009
0066 0066 0700 0200 0248 0AFF FFFF FFFF
FFFF FF54 0030 0030 0042 0042 0700 0100
0158 A000 8DA0 0083 FF
```

```
Type ICN# = GNRL
,128 (0)
.H
include MyIconFile
```



Programmer's Workshop

MPW Tools with TML Pascal II



TML Pascal II

Randy Leonard
Jacksonville, FL
Volume 5, Number 5

[Randy Leonard is currently employed by TML Systems, Inc. while completing his Master's thesis in computer science at the University of Central Florida. His thesis involves improving the efficiency of several key algorithms in constructive solid geometry (CSG). The thesis is written entirely with TML Pascal II and he uses several of his own MPW Tools to aid in his development.]

The introduction of the Macintosh Programmer's Workshop (MPW) v3.0 this January requires that we revisit this programming environment and study its new and exciting features as well as the significant features of previous versions of the product. This article discusses how the MPW environment can be enhanced with the addition of your own integrated programming tools. The type of commands introduced here are capable of executing in the background while running MPW v3.0. This capability enables you to continue your most important task in the foreground while such tasks as compiling and linking of programs occur in the background. Indeed, the tool introduced here may also run in the background.

Macintosh Programmer's Workshop

The Macintosh Programmer's Workshop is the official Macintosh development environment from Apple Computer, Inc. The MPW Shell is a complete development system for the Macintosh that includes, among other things, a multi-windowing text editor and a command processor. Also included with MPW is a linker, make facility, resource compiler, resource decompiler, source code management system, and more. There is also complete online help as well as an optional graphical interface for just about every command available in MPW.

MPW has often times been accused of having a steep learning curve. The author believes this false accusation can be based upon two pretexts. The first is due to unfamiliarity with the product. The second is due to the extraordinary power that MPW can provide its advanced users via the command line environment and MPW's advanced command processor. The truth of the matter is that MPW is as easy to learn and use as any other programming environment if you were to utilize only those features found in MPW that the other programming environments provide.

The true beauty of MPW is found in its open architecture. That is, anyone can expand the functionality of MPW to suit his own needs. There exist two methods of adding new commands to MPW: scripts and tools. This two part article will demonstrate how to create new programming tools and how to fully integrate these tools into the MPW environment. This month's article will give the motivation for writing your own MPW tool and show how it is done. Part two, to appear next month, will demonstrate how to integrate the tool into the MPW's online help facility and

how to develop a graphical interface for the new command.

MPW Tools

Tools provide a method of adding commands that are not inherent to the MPW command processor. For example, the TML Pascal II compiler is an extension to the MPW environment and is therefore implemented as a tool. Other examples of MPW Tools are the canonical speller Canon and TMLPasMat. TML Systems, in fact, used these two tools to put the finishing touches on both its Source Code Library II and example programs found on the TML Pascal II distribution disk. Canon adjusts all identifiers in source code files so they have the same capitalization as found in *Inside Macintosh*. TMLPasMat will format all Pascal source code files to a consistent format that you define. These two tools were especially helpful since programming styles of the programmers at TML Systems vary.

However, we needed even more help in providing a consistent format for all source code files that left our office. As it turns out, some programmers at TML Systems prefer different fonts and different font sizes for their work. Some programmers would print their work on the laser writer reduced to 75% while others would print at 100%. Some prefer different tab settings and still others have large screens on which to edit their programs. This last problem could be especially annoying to our customers with small screens since only a small part of the window of a text file would appear on their screen when they opened the file. Each customer could easily resolve the problem by resizing the window for his screen, but what if the window appeared off-screen to begin with!

Another problem unrelated to each programmer's taste was the problem of USYM resources. The TML Pascal II compiler writes symbol table information for separately compiled units to the resource fork of the main unit source code file as USYM resources. This is perhaps a more favorable solution than creating separate files to hold the information as other compilers do. Deleting a file that contains symbol table information is easy, but how do you easily erase resources from the resource fork of a source code file?

To solve the above problems, we developed a MPW tool that changed the font of MPW text files to any specified font and font size. Tab spacing is set to any defined value and the window is made to appear on any Macintosh screen with the top left corner slightly staggered from other windows and the bottom right corner extending to the bottom right of the screen. The page setup is reset to print at 100%. All USYM and other resources are optionally deleted as well. We call this tool ChangeTextRes. Below, we explain how to implement this tool, but first a little background.

What Every Tool has in Common

MPW Tools are usually invoked from the command line of the MPW Shell. The command line of an MPW command is simply the line on which the MPW command exists. Parameters are passed to the tool by placing them on the command line. A graphical interface for the tool, called the Commando, does exist, but is not typically used. The Commando interface for a tool is invoked by typing in the command name followed by the ellipses character (...). This character is obtained by holding the option key and pressing the semicolon. The implementation of a custom Commando interface for ChangeTextRes is discussed in next month's article.

Apple has defined several conventions for MPW Tools that allow them to work well together in an integrated fashion. First and most important is that a tool have some default behavior. Deviations from this default result only from options specified on the command line. A command line option is simply a parameter found on the command line starting with the character '-'. For example, the resource decompiler tool DeRez is invoked by typing:

```
DeRez filename
```

where filename specifies any resource file at all. If you wish only to decompile only dialog resources, you would type:

```
DeRez filename -only DLOG
```

Now, only resources of type 'DLOG' will be decompiled. The behavior of the DeRez tool has been modified by the -only option. Many command line options may be specified, but the order in which they are given should never matter.

Another rule is that tools are to run silently, they should require no interaction with the user to carry out its task. The only visual feedback from the tool should be in the form of a spinning cursor. The reasoning for these rules have their roots in the advanced capabilities of the MPW command processor. Should the reader wish to become a more powerful user of MPW, he is referred to the chapter "Using the Command Language" of the *Macintosh Programmer's Workshop Reference*.

The Command Processor

Each time a command is entered, the MPW command processor attempts to interpret it. If it is unsuccessful, the command processor assumes the command is either a tool, script, or an external application. Since we are writing a tool, we do not need to concern ourselves too much with the command processor, but there are a few features that need to be discussed.

Before invoking an MPW tool, the command processor first attempts to interpret parts of the command line. Certain characters have special meaning to the command processor. For example, the ~ (Option-x) character is a wild card character. If ~.p is specified, the command processor will expand this to a sequence of filenames that end with .p. The MPW Tool never sees the ~.p parameter. Rather, it sees the sequence of filenames

that match the ~.p pattern! This not only increases a user's productivity, but also simplifies the writing of tools.

Also allowed are special characters for I/O redirection, piping, and substitution of MPW Shell variables. Knowledge of these features are not necessary for the average MPW user. See the *Macintosh Programmer's Workshop Reference* for more details.

Three file variables are predefined for MPW Tools, input (standard input), output (standard output), and diagnostic. As expected, standard input is the Macintosh keyboard. Standard output is the current topmost window found in the MPW environment as is diagnostic.

How ChangeTextRes Works

The underlying theory of ChangeTextRes is quite simple. The tool receives from the command line the name of each file it is to work on as well as any command line options that may exist. The resource fork of each MPW text file specified on the command line is deleted if the command option -d is also present on the command line. If the -d option is not specified then no resources of any files are ever deleted. ChangeTextRes will then change the file's font, font size, and tab setting.

If ChangeTextRes is instructed to delete the entire resource fork with the -d option, there will no longer be any resource specifying the page setup or window size and position. If a file has no resource for page setup or file window position, the MPW Shell will assume default values. The default value for window position is to stagger the window's top left corner and cover the rest of the screen. The -d option will delete all USYM resources created by the TML Pascal II compiler as well.

ChangeTextRes assumes default values of Monaco, 9 point, and 3 for the font, font size, and tab setting, respectively. The user may change these values with the command line options -f, -s, and -t. The -f option, followed by a font name instructs ChangeTextRes to use that font name. The -s option followed by an integer value instructs ChangeTextRes to use the specified font size. And the -t option followed by an integer value instructs ChangeTextRes to use the specified tab setting. An error message is generated if an invalid font, a font size less than 1 or greater than 127, or a tab setting less than 1 or greater than 24 is specified.

To use ChangeTextRes, type the command followed by all file names and options desired. Keep in mind the MPW Shell's filename expansion capabilities discussed above. For example:

```
ChangeTextRes MyProject.proj *.p  
-f Courier -s 12 -t 4 -d
```

will delete all resources of all MPW text files ending with a .p as well as the file MyProject.proj. The font for each file is set to Courier 9 point and the tab setting is set to 4.

Accessing the Command Line

Accessing the command line parameters from within a tool is quite simple. The IntEnv unit defines two global variables used to access the command line parameters: argv and argc. The argv

variable is a pointer to an array of pointers to strings. Argc tells how many parameters exist in the argv array. The argv array starts at element 0 and argc is always one greater than the actual number of parameters found on the command line. The expression argv^[argc] is always equal to nil. The first parameter of the argv array (argv^[0]^) is the name of the MPW tool itself and is useful for error message generation. All parameters to the tool are stored in argv^[1]^ to argv^[argc-1]^ . It is up to the programmer to read in command line parameters and to distinguish between regular parameters and command line options. The ChangeTextRes tool contains the procedure ReadCommandLine to accomplish this task.

```

procedure ReadCommandLine;
var
  argVIndex      : integer;
  arg             : Str255;
begin
  if argc = 1 then SyntaxError(9, '');
  argVIndex := 1;
  while argVIndex < argc do begin
    arg := argv^[argVIndex]^;
    if length(arg) <> 0 then
      if arg[1] = '-' then
        if length(arg) > 1 then
          HandleOption(arg, argVIndex)
        else SyntaxError(8, '');
      argVIndex := argVIndex + 1;
    end; { while }
  end;
end;

```

In this routine, argVIndex is used to traverse the command line parameters. Each time a command line option is found, the procedure HandleOption is called. HandleOption will read the option and set program global variables accordingly. Since options may require an additional parameter (e.g. the ChangeTextRes option -f, -s, and -t), HandleOption must have the ability to read the next parameter(s) on the command line and increment argVIndex accordingly. If an invalid command line option or invalid value corresponding to a valid option is found, HandleOption will generate an error and terminate the program.

Rewriting a File's Resource Fork

The sole purpose for ChangeTextRes is to rewrite part or all of a file's resource fork. Since it is intended to work only on source code files, the segment of the program that effects a file's resource fork must first check to see if the file is of type 'TEXT' and has a creator of 'MPS ' (the MPW signature). If this is so, ChangeTextRes will proceed to change the file's resource fork. Below is the segment of code that accomplishes this task.

```

if (finderInfo.fdType = 'TEXT') and
   (finderInfo.fdCreator = 'MPS ') then begin
  if gResDelete then begin
    anOSError := OpenRF(filename, vRefNum, ResRefNum);
    anOSError := SetEOF(ResRefNum, 0);
    anOSError := FSClose(ResRefNum);
  end;
  result := IEFAccess(filename, F_STabInfo, gTabSize);
  result := IEFAccess(filename, F_SFontInfo, arg);
end

```

The global variable gResDelete is affected by the -d option. If the -d option is present on the command line, gResDelete is set to true, otherwise it is set to false. Only when gResDelete is true will all the resources be deleted.

An MPW file's font, font size and tab setting are modified with calls to the IEFAccess function. This function is defined in the IntEnv unit. There are three parameters to IEFAccess. The first is the filename on which the routine is to operate. The second parameter defines which operation to perform, and the third provides a means of passing data to and receiving results from the IEFAccess routine.

Our first call to IEFAccess sets the tab setting of a file. The predefined constant F_STabInfo informs IEFAccess to set the tab of the specified file and the third parameter specifies the desired tab setting. The second call to IEFAccess sets the font and font size. The second parameter of IEFAccess is set to the predefined constant F_SFontInfo. Apple has inadvertently documented that the third parameter, in this case, is a pointer to the new font and font size. This is not the case, rather, the upper word of this long integer is the font number and the lower word contains the font size. Both F_STabInfo and F_SFontInfo are defined in the IntEnv unit.

Spinning the Beach Ball Cursor

MPW Tools are, by convention, supposed to provide visual feedback to the user by displaying and spinning a cursor. By default, this cursor is the MPW beach ball cursor but may be any other cursor the programmer defines. Spinning cursors have a resource type of 'acur' and may be created with MPW's resource editor and/or resource compiler. See the appendix "Programming for the Shell Environment" in the *MPW Pascal Reference Manual* or Appendix D of the *TML Pascal II Language Reference* for more details on creating and using such resources.

All the routines related to the operation of the cursor by an MPW Tool are contained in the CursorCtl unit. Only two routines in this file are required by most tools: InitCursorCtl and RotateCursor.

For a tool to rotate the cursor, it must first call InitCursorCtl. This routine has just one parameter which is a handle to an 'acur' resource. If this parameter is nil, then the cursor defaults to MPW's spinning beach ball cursor. Call this routine very early in the program to prevent fragmentation of the heap.

Initializing the spinning cursor does not in itself cause the cursor to spin. The MPW Tool must manually spin the cursor itself. This may seem to be an inconvenience, but it is actually to your advantage. For example, a user can track progress of the MPW Linker by watching which way the beach ball rotates. The linker has three phases, each change in phase is accompanied by a change in direction of rotation of the cursor.

To rotate the cursor, call RotateCursor(value) where value is some integer or long integer. Each time this procedure is called, value is added to an internal counter. When this counter is an even increment of 32, the beach ball is rotated. If value is positive, the cursor is rotated in the clockwise direction. If value is negative, the cursor is rotated in the counter-clockwise direction. It is important to call RotateCursor on a frequent basis. It

is usually best to place this procedure call in the main loop of the program.

Software Interrupts

MPW Tools should be capable of responding to software interrupts known as signals. Currently, only one type of signal exists and that is the command-period (.). Signals have the capability of preempting a tool or any other MPW command. Tools will automatically respond to a signal but it may be necessary at times to prevent a signal from preempting a tool. Several routines in the Signal unit allow a tool to control the effect a signal may have on it. ChangeTextRes does not in any way attempt to control a signal's effect.

The default actions taken by a tool in response to a signal are to close all open files, execute any installed exit procedures, and terminate the program. See the appendix "Programming for the Shell Environment" in the *MPW Pascal Reference Manual* or Appendix D of the *TML Pascal II Language Reference* for more details on how to install exit procedures. Also refer to these manuals for information on how to prevent or delay the effects of signals on MPW Tools.

Returning Status Results

There are basically three different conditions that cause ChangeTextRes to terminate. The first is normal termination and arises when ChangeTextRes has successfully completed processing its data. There are two abnormal termination conditions. One is due to invalid syntax of command line parameters and the other to the inability for the tool to successfully complete its task. In any case, when a tool returns control to the MPW Shell, it must inform the Shell of its termination condition. This is done by returning a status code.

Defined in the IntEnv unit is the procedure IEExit. This procedure has one parameter of type LongInt. When a tool is to terminate, either normally or abnormally, it should call IEExit. Note that IEExit actually terminates the program. The value of its parameter is returned to the MPW Shell as a status code which by convention is zero to signify normal completion and non-zero to indicate abnormal termination. ChangeTextRes returns 1 to indicate a syntax error and 2 to indicate other errors.

Further Reading

This article has addressed many of the issues involved in writing MPW Tools, but there still remains a significant amount of potential yet to be realized. The chapter "Writing an MPW Tool" of the *Macintosh Programmer's Workshop Reference* discusses all the issues of writing MPW Tools, and does so in much greater detail than presented here. The chapter "Building an Application, a Desk Accessory, or an MPW Tool" of the same reference describes the mechanics of building a tool, but this knowledge is not necessary if you are using the TML Project Manager.

Chapter 8 of the *TML Pascal II User's Guide* shows how to write MPW Tools as does *Programming with Macintosh Programmer's Workshop*, by Joel West. This second book is highly recommended for any MPW user. Appendix D of the

TML Pascal II Language Reference and appendix titled "Programming for the Shell Environment" of the *MPW Pascal Reference Manual* give complete descriptions of the interface files required to create MPW Tools.

Introduction To Commando Interface

An MPW tool is typically invoked from the command line. This method of accessing a tool is sometimes cumbersome, especially if you forget the command line options available for the given tool. The Commando offers an optional graphical interface to an MPW tool from which any capability of the tool may be accessed. This graphical interface is implemented as a resource of type 'cmdo' and any resource number. Each tool may have its own unique 'cmdo' resource, but such a definition is not required. If more than one 'cmdo' resource is specified in the resource fork of a tool, then the one with the lowest resource number is selected. The MPW tool ChangeTextRes discussed last month was defined without such an interface. In this half, the standard Commando interface and the various controls its dialogs support are described. The article closes with an example interface for the tool created last month.

The Standard Commando Interface

Most Commando dialogs share the same basic structure similar to the dialog for the MPW Search command seen in Figure 1. This dialog is divided primarily into three boxes. The first is the Options box. In this region are the controls used to access the various options of the command. There are numerous types of controls available to the designer of a Commando dialog, all of which are discussed in a later section. It is intended that every aspect of a given tool may be accessed via the controls in this box. Other dialog boxes, called subviews, may be accessed from button controls in this region to allow for a more organized and readable Options box.

The middle of the dialog contains what is called the Command box. The Command box is updated to reflect the current list of command line options each time a control in the Options box is modified. This command line may be copied into the clipboard using Command-C or the Edit menu. Below the Command box is the Help box. Clicking and holding down the mouse button on any control causes help information related to that control to be written in this box.

Located in the bottom right corner of the dialog are two buttons. These two buttons are generically referred to as the Cancel and Do It buttons. Clicking Cancel or pressing Command-period causes the Commando to be aborted. Clicking the Do It button or pressing the Enter key causes the command to be passed back to the MPW Shell for execution. Special results may be obtained, however, by holding down various modifier keys while pressing Enter. Holding the Option key while pressing Enter causes the command to be executed as well as echoed to MPW's active window. Alternatively, holding the Command key while pressing Enter causes no command to be executed, it just causes control to be passed back to the Shell. Holding both the Option and Command keys and pressing Enter results in the command line being written to the active window without being

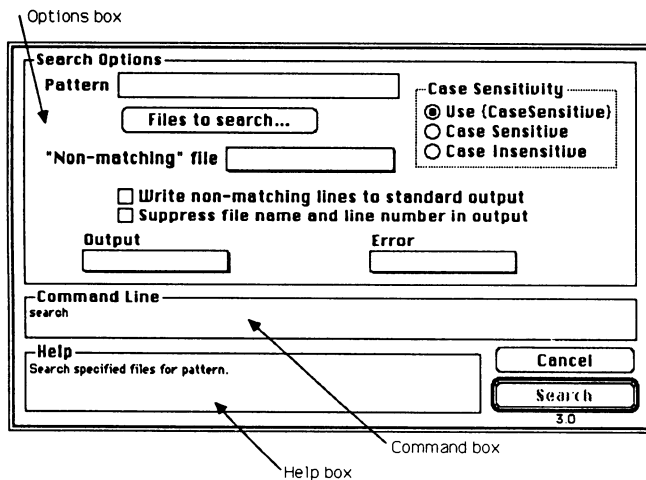


Figure 1

executed.

Invoking the Commando

There are three ways to invoke the Commando dialog. Perhaps the easiest is to type the name of the command and pressing Option-Enter, but this method is only available to users of MPW v3.0. Pressing Enter on a line containing the command name followed by the ellipsis (...) character will also activate the Commando. The ellipsis character is obtained by holding the Option key while pressing the semicolon (;) character. The third method is to type the word commando followed by the name of the tool to be invoked, and then pressing Enter.

The latter method deserves special attention. Commando is actually an MPW tool. Typing commando and pressing Enter will activate this tool, which will then read the 'cmdo' resource of the tool specified on the command line and display its dialog interface. This method of invoking the Commando results in only the generated command to be written to the topmost window, the command is never executed.

Standard dialog box controls

The 'cmdo' resource allows for various types of controls to be displayed in the Options box. Only controls supported by Commando may be displayed, but with the types of controls provided, any required functionality should be easily accomplished. The controls and other items that may appear in the Options box include, but are not limited to:

- Check boxes
- Radio buttons
- Boxes, lines, and text titles
- Pop-up menus
- Lists
- Three-state buttons
- Icons and Pictures
- Nested dialog boxes
- Redirecting output
- Multiple input files

- Multiple directories
- Single input or output file
- Version number

The chapter Using MPW: The Basics in the MPW Reference Manual contains a section detailing all the different types of controls allowed within a Commando dialog. The chapter titled Creating Commando Interfaces for Tools of the same reference gives complete resource descriptions for each type of control. Only a few types of controls and other Commando items will actually be described here.

Radio Buttons

Options that are mutually exclusive of each other are often presented as a group of radio buttons. The tab settings for ChangeTextRes are implemented as a cluster of radio buttons in the Commando interface defined here. Radio button groups are usually surrounded by a labeled perimeter to emphasize the grouping of the controls. The tab option was specified as a group of radio buttons in the ChangeTextRes Commando for example only. Depending upon taste, better methods probably exist for presenting this option.

A cluster of radio buttons is considered a single item in the resource description. The Rez type definition for a radio button cluster is given as follows:

```
case RadioButtons:
    key byte = RadioButtonsID;
    byte = $$CountOf(radioArray); /* # of buttons */
    wide array radioArray {
        rect; /* bounds */
        cstring; /* title */
        cstring; /* option returned */
        byte NotSet, Set; /* whether button is set or not */
        cstring; /* help text for button */
        align word;
    };
```

A general rule to follow when reading Rez type definitions such as this is that any field containing an equal sign can effectively be ignored by the programmer. Rez inserts values for such fields automatically at compile time. In this resource type definition, a cluster of radio buttons is defined to be an array of radio button elements, with each element consisting of an enclosing rectangle, title, corresponding command line option, default value, and a help field. A sample excerpt from the ChangeTextRes 'cmdo' resource is as follows:

```
NotDependent ( ), RadioButtons (
    /* array radioArray: 21 elements */
    /* [1] */
    (40, 30, 55, 60), "1", "-t 1",
    notSet, "Set tabs to one spaces",
    /* [2] */
    (60, 30, 75, 60), "2", "-t 2",
    notSet, "Set tabs to two spaces",
    /* [3] */
    (80, 30, 95, 60), "3", "-t 3",
    Set, "Set tabs to three spaces",
    /* [4] */
    (100, 30, 115, 60), "4", "-t 4",
    notSet, "Set tabs to four spaces",
```



```

...
/* [20] */
(140, 170, 155, 210), "20", "-t 20",
notSet, "Set tabs to twenty spaces",
/* [21] */
(160, 170, 175, 210), "21", "-t 21",
notSet, "Set tabs to twenty one"
"spaces",
)
/* array radioArray */
),
/* RadioButtons cluster */

```

Only one radio button should be marked as Set in the cluster, where the set button is defined as the default button. If no buttons are defined as set, then the first button in the list is assumed to be the default. If a particular radio button is defined to be the default, then its option return value (e.g. "-t 3") is not written to the command line when that option is selected.

Check Boxes

Options to MPW tools are often Boolean flags, that is, they are either off or on non-exclusively of other options. Such options are best represented in the Commando dialog with check boxes. A group of related check boxes may or may not be surrounded by a labeled perimeter. The Rez type definition for check boxes is as follows:

```

case CheckOption:
key byte = CheckOptionID;
byte NotSet, Set; /* whether button is set
or not */
rect; /* bounds */
cstring; /* title */
cstring; /* option returned */
cstring; /* help text for button */
);

```

This type definition is similar to that of the radio buttons. There is one instance of a check box control in the ChangeTextRes Commando dialog. It is used to set the flag for deleting all resources for selected files.

```

NotDependent ( ), CheckOption (
NotSet,
(100, 250, 115, 425),
"Delete resource fork",
"-d",
"Delete all resources for the selected"
"files"
),

```

Pop-Up menus

There are two types of pop-up menu controls allowed in a Commando dialog: shadow and editable. A shadow pop-up menu is a field displayed as a shadowed box. Clicking inside this box will cause a pop-up menu to appear containing all valid choices for that particular field. The shadowed box is aligned around the current selection in the menu, and this selection is checked as well. This menu will behave just like the standard pull-down menu for the duration of the time the mouse button is held down. The menu will scroll vertically if required and the

selected menu item is placed in the menu box when the mouse button is released. The font control of the ChangeTextRes Commando is a shadow pop-up menu. The Rez definition for this type of resource is:

```

case PopUp:
key byte = PopUpID;
byte Window, Alias, Font, Set;
/* popup type */
rect; /* bounds of title */
rect; /* bounds of popup line */
cstring; /* title */
cstring; /* option returned */
cstring; /* help text for popup */
byte noDefault, hasDefault;
/* hasDefault if first item
is "Default Value" */

```

A shadow pop-up menu may contain one of four possible lists. The only one we are concerned with here is Font. If Font is selected as the pop-up type, then all available fonts are displayed in a scrollable pop-up menu. Also specified in this resource type definition are enclosing rectangles of the shadow box, the control's title, enclosing rectangle of the control's title, option returned, and help information. Also, if the last item in the resource definition is set to hasDefault, then the first item in the scrollable menu is "Default Font".

The shadow pop-up menu is convenient, but not always sufficient. If the desired value is not available in the menu box, then some means should exist for it to be manually entered. The type of control that allows a pop-up menu and edit box combined in one is called an editable pop-up menu. The Font Size control in the ChangeTextRes Commando is such a control. The resource type definition for this control is as follows:

```

case EditPopUp:
key byte = EditPopUpID;
byte MenuTitle, MenuItem,
FontSize, Alias, Set;
/* Type of editable pop-up */
rect; /* bounds of title */
rect; /* bounds of text edit area */
cstring; /* title */
cstring; /* option to return */
cstring; /* help text for textedit part */
cstring; /* help text for popup part */

```

This definition is almost identical to that for shadow pop-up menus. As with shadow pop-up menus, only font information is discussed. When FontSize is selected as the type of menu, only the available font sizes for a specified font are listed. If none of the font sizes in the menu are desirable, then the user may type in his own value.

Of special importance here is the association of this pop-up menu with some font. Editable pop-up menus are required to be made dependent on another shadow pop-up menu containing a list of fonts so that the control knows what font sizes to display. Dependent controls are not discussed in this article, but one such control is actually implemented. The excerpt from the ChangeTextRes 'cmdo' resource implementing two pop-up menus with one dependent upon the other is given below.

```

/* [3] */
NotDependent ( ), PopUp (
    Font,
    (20, 250, 40, 320),
    (20, 330, 40, 450),
    "Font",
    "-f",
    "Select font specified files are"
    " to assume",
    hasDefault
),
Or ((1)), EditPopUp (
    FontSize,
    (50, 250, 70, 320),
    (50, 330, 70, 400),
    "Font Size",
    "-s",
    "User definable font",
    "Select from available fonts"
),
),

```

All that will be said for this instance of dependent controls is that the editable pop-up menu is made dependent on the shadow pop-up menu. If no font is selected in the shadow pop-up menu, then the pop-up menu part of the editable pop-up menu is disabled. The editable part of the control, however, is still active.

Multiple Input Files

When a tool allows more than one input file of a given type, then a single button control is displayed. An example of this would be the Files... button in Figure 2. Clicking this button will cause a modified version of the standard SGetFile dialog to appear. This is shown in Figure 3. This dialog allows multiple files from various directories to be selected. All selected files are listed in the lower scrollable list. Files may be selectively removed from this list. A file filter may be applied to all files displayed in the upper scrollable list.

Modifying Existing Commandos

When creating any standard type of graphical resource, MPW users usually prefer ResEdit. But when it comes to fine tuning, the resource is decompiled to text format, modified by hand, and then compiled back to object code with the Rez compiler. ResEdit, though, does not have the capability to graphically create and edit Commando interfaces. Instead, the Commando tool for MPW v3.0 has a built in editor that allows the programmer to edit text labels and help messages. Controls and other items may be moved and resized with this editor as well. Unlike ResEdit, controls cannot be created, duplicated or deleted.

The process of creating Commando resources is to first create the resource text file. This may be done with little concern for where controls are actually placed in the dialog because resources may then be compiled to object code format with the MPW tool Rez. The dialog is then modified with the Commando editor. The modified interface may then be decompiled back to text for fine tuning using the DeRez tool. Figure 4 illustrates the interrelationships of the various tools that may be used in creating

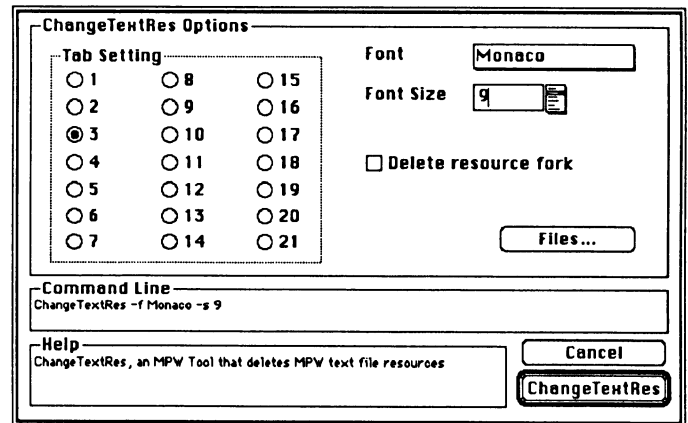


Figure 2

a Commando interface.

Commando's built in editor may be activated by holding down the Command key immediately after launching Commando until the Watch cursor appears. This editor may also be invoked from the command line by adding the -modify option in one of the following ways:

```
Commando ChangeTextRes -modify
```

or

```
ChangeTextRes... -modify
```

Commando may be used in two different modes once it has been launched with the built-in editor enabled. The first is Normal mode in which the interface operates as usual. The second is the Edit mode in which controls may be moved and resized. To activate the Edit mode, hold the Option key down while selecting a control. Continue to hold the Option key down while the control is selected to drag and/or resize the control.

The Commando editor works in many ways like the ResEdit graphical editor. It also allows multiple controls to be simultaneously selected and moved. Text labels may be edited much in the same way the text for an icon is changed from the Finder.

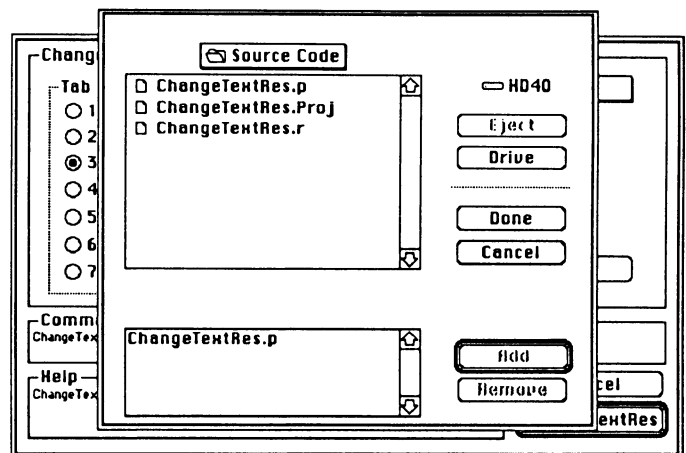


Figure 3

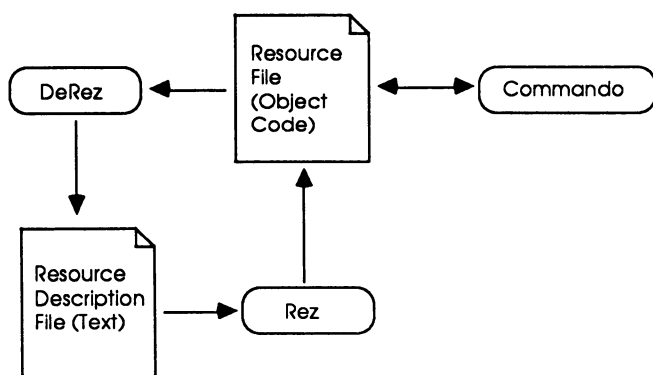


Figure 4

Help messages as well as the size of the Commando dialog box may be modified as well.

If the Commando's Edit mode was activated at any time, then selecting the Cancel or Do It button to exit the dialog causes a Save dialog to appear. This Save dialog offers three options: save the resource, do not save the resource, or cancel the save operation and return to the Commando dialog. Saving a Commando resource causes the original resource to be overwritten, so caution is advised.

The ChangeTextRes Commando

The resource file given at the end of this article defines the Commando interface illustrated in Figures 2 and 3. They were created using the techniques described in this paper. First, the resource text file was entered and compiled with Rez. Next, the Commando editor was invoked to align and resize all controls. The DeRez tool was then used to decompile the 'cmdo' resource back to the text given here. Assuming the filename containing the resource text file is called ChangeTextRes.r, the Rez command used to compile this resource is:

```
Rez ChangeTextRes.r cmdo.r -append
-o ChangeTextRes
```

The file cmdo.r contains the resource type declaration for the resource type 'cmdo'. This file comes with MPW and the Rez compiler knows where to find it. Cmdo.r may be omitted from this command if it is specified as an include file within the ChangeTextRes.r file. After the Commando editor has been used to move and resize fields, the following DeRez command may be used to obtain a text file containing the modified 'cmdo' resource.

```
DeRez ChangeTextRes cmdo.r -only 'cmdo'
```

Of course, instead of using the MPW command line, the Commando interface for the Rez and DeRez tools could have been used. Just think, using Commando to create new Commando interfaces!

Conclusion

Commando dialogs offer an optional graphical interface to MPW commands implemented as tools. They offer a means to

graphically select any feature of the MPW command they represent and may result in that command being executed. Creating a Commando interface is as easy as creating a resource of type 'cmdo' using Rez, DeRez, and Commando. ResEdit is not capable of editing 'cmdo' resources, but since Commando has its own built in editor, it does not matter.

```
program ChangeTextRes;
(
    ChangeTextRes.p

    An MPW Tool to delete resource fork of MPW
    text files and rewrite the resource fork
    to specify a desired tab setting, font,
    and font size.

    (c) TML Systems, Inc., 1988
    All rights reserved.
)

uses MemTypes, QuickDraw, OSIntf, ToolIntf,
    PackIntf, PasLibIntf,
    ( required for MPW Tools )
    CursorCtl, IntEnv;

var
    ResRefNum : integer;
    ( reference number for resource fork of a given file )
    filename : Str255;
    aStringPtr : StringPtr;
    ( reference number for default drive )
    vRefNum : integer;
    ( Finder information for a given file )
    fndrInfo : FInfo;
    ( result from Mac ROM file I/O calls )
    anOSErr : OSErr;
    ( passed to IEFAccess specifies font and font size )
    arg : LongInt; ( result from IEFAccess calls )
    result : LongInt;
    i : integer;

    ( Font number of specified font as returned by GetFNum )
    gFont : integer;
    gFontSize : LongInt;
    gTabSize : LongInt; ( tab setting )
    ( delete all of file's resources? )
    gResDelete : boolean;

function UpperCase(str: Str255): Str255;
(
    Convert an alphanumeric string to all
    uppercase characters.
)

var
    i: integer;
begin
    for i := 1 to length(str) do
        if (str[i] >= 'a') and
            (str[i] <= 'z') then
            str[i] := chr(ord(str[i]) - 32);
    UpperCase := str;
end;

procedure SyntaxError(err: integer; msg: Str255);
(
    Display the appropriate syntax error and
    then exit from the program. Return a
    status value of 1 indicating an early
    termination of program.
)
begin
    case err of
```

```

1: writeln('# ', msg, ' is an invalid option');
2: writeln('# missing font');
3: writeln('# missing font size');
4: writeln('# missing tab setting');
5: writeln('# ', msg, ' is an invalid font');
6: writeln('# ', msg, ' is an invalid font size');
7: writeln('# ', msg, ' is an invalid tab size');
8: writeln('# the - character must be
   accompanied by an option');
9: begin
    writeln('# Usage - ChangeTextRes [name...] ');
    writeln('    -f fontname      # set
      font of files to fontname');
    writeln('    -s fontsize     # set
      font size of files to fontsize');
    writeln('    -t tabs         # set
      tab setting to tabs');
  end;
otherwise
  writeln('fatal error #', err);
end;
IEExit(1); { return error status of 1 }
end;

```

```

procedure HandleOption(opt: Str255;
  var argIndex: integer);

```

```

(
  Set the appropriate global flag for each
  command line option encountered on the
  command line. If an invalid
  option is found, give an error message and
  exit from the program. If the option
  requires an additional command line
  parameter (e.g. -f Monaco), then retrieve
  the option(s) needed and increment the
  argIndex counter appropriately.
)

```

```

var
  NumString, str      : Str255;

```

```

begin
  str := UpperCase(opt);
  Delete(str, 1, 1); {delete the '-' character}
  if str = 'F' then begin { set font }
    argIndex := argIndex + 1;
    if argIndex < argc then begin
      GetFNum(argv[argIndex], gFont);
      if gFont < 0 then
        SyntaxError(5, argv[argIndex]);
    end
    else SyntaxError(2, '');
  end
  else if str = 'S' then begin { set font size }
    argIndex := argIndex + 1;
    if argIndex < argc then begin
      StringToNum(argv[argIndex], gFontSize);
      if (gFontSize <= 0) or
        (gFontSize >= 128) then begin
        NumToString(gFontSize, NumString);
        SyntaxError(6, NumString);
      end;
    end
    else SyntaxError(3, '');
  end
  else if str = 'T' then begin { set tab }
    argIndex := argIndex + 1;
    if argIndex < argc then begin
      StringToNum(argv[argIndex], gTabSize);
      if (gTabSize <= 0) or
        (gTabSize >= 25) then begin
        NumToString(gFontSize, NumString);
        SyntaxError(7, NumString);
      end;
    end;
  end;
end;

```

```

    end
  else SyntaxError(4, '');
end
else if str = 'D' then
  gResDelete := true
else SyntaxError(1, str);
end;

```

```

procedure SkipOption(opt: Str255; var argIndex: integer);

```

```

(
  This routine is called only after the
  command line parameters have already been
  scanned once using HandleOption. The
  purpose of this routine is to properly
  increment argIndex according to the
  appropriate command line options.
)

```

```

var
  str: Str255;

```

```

begin
  str := UpperCase(opt);
  Delete(str, 1, 1); {delete the '-' character}
  if str = 'F' then { set font }
    argIndex := argIndex + 1
  else if str = 'S' then { set font size }
    argIndex := argIndex + 1
  else if str = 'T' then { set tab size }
    argIndex := argIndex + 1
  else if str = 'D' then { nothing }
  end;
end;

```

```

procedure ReadCommandLine;

```

```

var
  argVIndex      : integer;
  arg             : Str255;

```

```

begin
  if argc = 1 then SyntaxError(9, '');
  argVIndex := 1;
  while argVIndex < argc do begin
    arg := argv[argVIndex];
    if length(arg) <> 0 then
      if arg[1] = '-' then
        if length(arg) > 1 then
          HandleOption(arg, argVIndex)
        else SyntaxError(8, '');
      argVIndex := argVIndex + 1;
    end; { while }
  end;
end;

```

```

procedure ReportError(error: integer; filename: Str255);

```

```

(
  Generate the appropriate error message
  then exit from the program. Return a
  status value indicating early termination
  from the program.
)

```

```

begin
  if error = 0 then
    exit(ReportError);

  write(diagnostic, 'ERROR! ');
  case error of
    -35: writeln(diagnostic, filename, ' volume does not exist');
    -36: writeln(diagnostic, filename, ' IO Error');
    -37: writeln(diagnostic, filename,
      ' is a bad filename or volume name');
    -42: writeln(diagnostic, 'Too many files open');
    -43: writeln(diagnostic, filename, ' not found');
    -45: writeln(diagnostic, filename, ' is locked');
  end;
end;

```

```

-46: writeln(diagnostic, filename,
      ' is locked by a software flag');
-47: writeln(diagnostic, filename,
      ' is busy; one or more files are open');
-53: writeln(diagnostic, filename, ' volume not on-line');
-54: writeln(diagnostic, filename,
      ' cannot be opened for writing, file is locked');
-61: writeln(diagnostic, filename,
      ' Read/write permission doesn't allow writing');
otherwise
  writeln(diagnostic, 'OS error #',
    error, ' has occurred. ');
  writeln(diagnostic, ' Reference Inside
    Macintosh pp. III:205-209 for further details');
end;
IEExit(2);
end;

begin (main program)
  ( make first stmt to avoid heap fragmentation )
  InitCursorCtl(nil);
  InitFonts; (so we can read in font names)

  ( so we read in JUST the font names! )
  SetResLoad(false);

  ( Set default values )
  gResDelete := false;
  gFont := 4;
  gFontSize := 9;
  gTabSize := 3;

  ReadCommandLine;

  arg := gFont;
  arg := BSL(arg, 16);
  arg := arg + gFontSize;

  anOSError := GetVol(aStringPtr, vRefNum);
  if anOSError <> 0 then
    ReportError(anOSError, aStringPtr);

  i := 1;
  while i < argc do begin
    ( Make cursor rotate each time through loop )
    RotateCursor(32);
    filename := argv[i];

    if length(filename) = 0 then begin
      i := i + 1;
      cycle;
    end;

    if filename[i] = '-' then
      SkipOption(filename, i)
    else begin ( valid filename )

      anOSError := GetFInfo(filename, vRefNum, fnderInfo);
      if anOSError <> 0 then begin
        ReportError(anOSError, filename);
        cycle;
      end
    else begin ( file exists )
      if (fnderInfo.fdType = 'TEXT') and
        (fnderInfo.fdCreator = 'MPS ')
        then begin
        if gResDelete then begin
          anOSError := OpenRF(filename, vRefNum, ResRefNum);
          anOSError := SetEOF(ResRefNum, 0);
          anOSError := FSClose(ResRefNum);
        end;
        result := IEFAccess(filename, F_STabInfo, gTabSize);
        result := IEFAccess(filename, F_SFontInfo, arg);
      end
    end
  end
end

```

```

else
  writeln('WARNING! ', filename, ' is not an MPW text
file, resources not deleted');
end; ( file exists )

end; ( valid filename )
i := i + 1;
end; ( while i < argc )
writeln;

SetResLoad(true);
IEExit(0); ( Normal status return )
end. (main program)

/*
ChangeTextRes.r
This file contains only one resource which
defines the commando interface for the MPW
Tool ChangeTextRes
*/
#include "cmdo.r"

resource 'cmdo' (128, "ChangeTextRes") (
  ( /* array dialogs: 2 elements */
    300,
    "ChangeTextRes, an MPW Tool that" deletes
    "MPW text file resources",
    ( /* array itemArray: 3 items */
      /* [1] */
      NotDependent ( ), RadioButtons (
        ( /* array radioArray: 21 elements */
          /* [1] */
          (40, 30, 55, 60), "1", "-t 1",
          notSet, "Set tabs to one spaces",
          /* [2] */
          (60, 30, 75, 60), "2", "-t 2",
          notSet, "Set tabs to two spaces",
          /* [3] */
          (80, 30, 95, 60), "3", "-t 3",
          Set, "Set tabs to three spaces",
          /* [4] */
          (100, 30, 115, 60), "4", "-t 4",
          notSet, "Set tabs to four spaces",
          /* [5] */
          (120, 30, 135, 60), "5", "-t 5",
          notSet, "Set tabs to five spaces",
          /* [6] */
          (140, 30, 155, 60), "6", "-t 6",
          notSet, "Set tabs to six spaces",
          /* [7] */
          (160, 30, 175, 60), "7", "-t 7",
          notSet, "Set tabs to seven spaces",
          /* [8] */
          (40, 100, 55, 140), "8", "-t 8",
          notSet, "Set tabs to eight spaces",
          /* [9] */
          (60, 100, 75, 140), "9", "-t 9",
          notSet, "Set tabs to nine spaces",
          /* [10] */
          (80, 100, 95, 140), "10", "-t 10",
          notSet, "Set tabs to ten spaces",
          /* [11] */
          (100, 100, 115, 140), "11", "-t 11",
          notSet, "Set tabs to eleven spaces",
          /* [12] */
          (120, 100, 135, 140), "12", "-t 12",
          notSet, "Set tabs to twelve spaces",
          /* [13] */
          (140, 100, 155, 140), "13", "-t 13",
          notSet, "Set tabs to thirteen spaces",
          /* [14] */
          (160, 100, 175, 140), "14", "-t 14",
          notSet, "Set tabs to fourteen spaces",
          /* [15] */

```

```

    {40, 170, 55, 210}, "15", "-t 15",
notSet, "Set tabs to fifteen spaces",
/* [16] */
    {60, 170, 75, 210}, "16", "-t 16",
notSet, "Set tabs to sixteen spaces",
/* [17] */
    {80, 170, 95, 210}, "17", "-t 17",
notSet, "Set tabs to seventeen spaces",
/* [18] */
    {100, 170, 115, 210}, "18", "-t 18",
notSet, "Set tabs to eighteen spaces",
/* [19] */
    {120, 170, 135, 210}, "19", "-t 19",
notSet, "Set tabs to nineteen spaces",
/* [20] */
    {140, 170, 155, 210}, "20", "-t 20",
notSet, "Set tabs to twenty spaces",
/* [21] */
    {160, 170, 175, 210}, "21", "-t 21",
notSet, "Set tabs to twenty one spaces",
} /* array radioArray */
), /* RadioButtons cluster */

NotDependent ( ), TextBox (
    gray,
    {30, 20, 185, 220 },
    "Tab Setting"
),

/* [3] */
NotDependent ( ), PopUp (
    Font,
    {20, 250, 40, 320},
    {20, 330, 40, 450},
    "Font",
    "-f",
    "Select font specified files are to"

```

```

    " assume",
    hasDefault
),
Or ({1}), EditPopUp (
    FontSize,
    {50, 250, 70, 320},
    {50, 330, 70, 400},
    "Font Size",
    "-s",
    "User definable font",
    "Select from available fonts"
),
NotDependent ( ), MultiFiles (
    "Files...",
    "Select files who's resources are to"
    " be deleted",
    {155, 350, 175, 450},
    "",
    "",
    MultiInputFiles (
        {text},
        "",
        "",
        ""
    )
),
NotDependent ( ), CheckOption (
    NotSet,
    {100, 250, 115, 425},
    "Delete resource fork",
    "-d",
    "Delete all resources for the selected"
    "files"
)
) /* array itemArray */
} /* array dialogs */
);

```



Advanced Mac'ing

The Notification Manager



Steven Sheets
Contributing Editor
Haufman Estates, IL
Volume 5, Number 5

Hey You: The Mac Notification Manager

More and more, code is being written for the Macintosh that runs in a multitasking environment. At first, an application only had to share space with Desk Accessories. Then came MultiFinder, and multiple applications started running at the same time. Along the way, Macintosh programmers started creating various low level routines to do tricks with the machine (INITs that load routines into system memory, vertical retrace routines, time manager 'wakeup' routines, redirection of normal traps, etc.). At any given time, half a dozen or more code segments may be running in addition to a single 'normal' application.

These code segments have to work in a very restrictive environment. Some are not allowed access to global data. Many can not make Toolbox calls that move or purge memory. Others can not even have control of the CPU for longer than a tick (60th second). Even a normal application cannot place an alert where the user can see it if it is running in the background under MultiFinder. How can these code segments inform the user of some event or action with these limitations?

Enter one of newest Toolbox managers from Apple; the Notification Manager. The Notification Manager allows the code to post a notification request that, in some method, notifies the user of something. A Notification Request can consist of an Alert Notification (displaying a text message), an Audible Notification (playing a 'snd' sound), an Icon Notification (displaying a small icon, or 'SICN', in the upper left corner on top of the Apple icon) or some combination.

Using the Notification Manager, a terminal program running in the background can inform the user when a download is complete. No matter what is running at the time, when the request is processed, the Notification Alert will appear on top of all windows. Using the Manager, an Appletalk Mail program could inform the user of new mail with a flashing mail box icon.

The Notification Manager handles the processing of all requests (playing the sound, clicking on the alert, flashing the icon, etc.) asynchronously. The code segment would use two new Toolbox traps to inform the manager of a request. Neither of these two calls process the request, they simply add or detach records from the Notification Manager list. Because of this, they neither move nor purge memory, and can be safely called at any time, even inside of a low level interrupt procedure.

The Notification Manager was implemented in the System Software 6.0. The new Toolbox traps will not function on systems prior to 6.0. Portions of the System Software 6.0 already uses the Notification Manager. By flashing a small Clock icon, The Alarm Clock use the manager to signal that the alarm has gone off. The Print Monitor also uses the Notification Manager to signal printing information.

The Notification Manager

The Notification Manager maintains a queue (list) of requests that need to be processed. Whenever SystemTask or WaitNextEvent is called, the Notification Manager checks the queue for an unprocessed request. First the Notification Request, if there is one, is processed, then the response routine, if there is one, is called. The response routine can be the default routine (simply removes the request record from the queue) or the code segment can define it's own response routine to do whatever it wants.

The Notification Request is a standard Macintosh Queue record expanded for the Notification Manager. The record has the following format:

```
NMRec = record
  qLink: QElemPtr;
  qType: INTEGER;
  nmFlags: INTEGER;
  nmPrivate: LONGINT;
  nmReserved: INTEGER;
  nmMark: INTEGER;
  nmSIcon: Handle;
  nmSound: Handle;
  nmStr: StringPtr;
  nmResp: ProcPtr;
  nmRefCon: LONGINT;
end;
```

The qLink pointer points to the next request in the linked list and should not be changed by the code segment. Neither should the nmPrivate or nmReserved elements be changed by the code; the manager handles them. The qType (type of queue) of a Notification Queue is the value 8, the integer value of ORD(nmType). The qType element needs to be set to 8 before the request is added to the queue.

The nmStr pointer points to the text message that would be displayed by the Notification Manager in an alert. If the pointer is set to nil, no alert will be displayed. The nmSound is a handle to a sound record (ie. 'snd' resource). The Sound Manager will play it before the alert is displayed. If nmSound is set to -1, the "System Error" sound is used. If it is set to 0, no sounds are played. The nmSIcon is the handle to the small icon (32 bytes, 16 by 16 pixel bitmap) that the Notification Manager will flash in the upper left corner of the screen on top of the Apple icon. Usually this small icon is stored as a resource of type 'SICN'. A nil value for nmSIcon signifies no flashing icon. The nmMark element indicates who generated the notification event. An application should set nmMark to 1. When under MultiFinder, a diamond mark will be placed in the Apple menu next to the application's name. If the request was generated by a Desk

Accessory, nmMark should be set to the refNum of the DA. This will place a diamond mark next to the DA's name. All other types of code segments (Drivers, patches, etc.) should set this value to 0.

The nmResp is the pointer to the procedure called after the request is processed. If nmResp is set to 0, no procedure is called. If it is set to -1, a special response routine is called that removes that request from the queue. This response routine does not dispose of any memory (the string, the sound, the icon, or the request itself). If the code segment wants to define its own response routine, nmResp is set to the pointer of that routine. In most cases (main exception is Icon Alerts), the response procedure should remove the request from the queue immediately. The response procedure has the following format:

```
PROCEDURE MyResponse (nmReqPtr : QElemPtr) ;
```

The response routine will be called by the Notification Manager during a SystemTask or WaitNextEvent call. Thus it is safe to move or purge memory. It is also safe to do any I/O (file, resource, serial, Appletalk). If the routine is suppose to change some global variables of an application, it must first make sure that the A5 is correct. For example, when under MultiFinder, the notification may occur when some other application's A5 is set. The nmRefCon may come in handy to store the value of A5. The nmRefCon element can be used by the code segment for whatever purpose the code segment wants. A handle to a data structure, a pointer to a Boolean flag, or the correct A5 value could be stored there.

Apple states that the response routine should not do "user interface" type of drawing. Instead the routine should set a flag so that the application knows when it should do the drawing. To change a Boolean flag, it is simpler to stuff a pointer to the flag into nmRefCon than it is to change A5 back and forth.

Using the Notification Manager

There are only two routines that exercise the Notification Manager, NMInstall and NMRemove. NMInstall is passed the pointer to the notification request that is to be installed into the queue. It returns noErr (0) if successful. NMRemove is passed a pointer to a notification request already installed in the queue. If it is successful in removing that request, it returns noErr also. Unless a request wants an Icon to be flashed for awhile, the NMRemove routine should be called after a notification request has been processed. Both calls are register based. The following are the Inline Glue routines needed to use the call from Pascal and the Assembler information:

```
FUNCTION NMInstall (nmReqPtr : QElemPtr) : OSErr;
    inline $205F, $A05E, $3E80;
```

```
_NMInstall ($A05E)
On entry: A0 pointer to the NMRec
On exit: D0 result code
```

```
FUNCTION NMRemove (nmReqPtr : QElemPtr) : OSErr;
    inline $205F, $A05F, $3E80;
```

```
— NMRemove ( $A05F )
On entry: A0 pointer to the NMRec
On exit: D0 result code
```

The Sample Program: HeyYou

HeyYou uses the Notification Manager to show 3 different effects. The NotifStr routine demonstrates the simplest effect, displaying a text message to the user. The routine uses two global variables, a string to hold the text, and a NMRec to hold the notification request. The request nmResp is set to -1 so that the Notification Manager will automatically remove this request from the queue after it has been done. Since the data is stored in global variables, no other clean up has to be done.

The NotifSnd routine demonstrates a slightly more involved request. It plays a sound ('snd' resource) and displays a text message. No global variables are used in this example. Instead memory is allocated to hold the request and the test string. The nmResp is set to a pointer to the MySndResponse so that when the request is done, the MySndResponse routine can be called. MySndResponse can remove the request from the queue, release the sound from memory, dispose of the memory holding the string, and finally dispose of the memory holding the request itself.

The method used in NotifSnd works well for code segments that have no access to global memory space such as Desk Accessories, INITs, etc. This method also has the additional advantage of allowing more than one request to be added at the same time. This could not be done with NotifStr since each request shared the same global variables. As long as MySndResponse is locked in memory somewhere, the method used in NotifSnd will work.

The last example, NotifIcon, demonstrates the use of icons by the Notification Manager. It displays a text message, blinks a small icon in the upper left corner of the screen and marks HeyYou as the application that caused this request. Usually the code segment wants the icon to flash in the corner until the user has done something in response. For example, the Alarm Clock does not remove its icon until the alarm is turned off. To keep the Icon in use until the appropriate time, the notification request can not be removed from the queue or the flashing effect would be lost. Thus no response routine is specified. Instead the application removes the request (and flashing icon) when the application quits, when it resumes after a switch under MultiFinder, or when the user selects the "Remove Icon" menu item. In HeyYou, one example sets a timer going so that NotifIcon is processed 60 seconds after the menu item is selected. Since HeyYou and the timer work in the background under MultiFinder, this will demonstrate how the Notification Manager works when some other application is running. Notice the mark that appears next to the program name HeyYou in the Apple menu.

Beyond...

Here are some last ideas for Notification Manager routines/utilities that can be written:

1) An example that places a Notification Request in low System memory, and starts a Timer routine that will install the

request sometime in the future. The response routine is set so that it removes the request and releases the memory used by everything (string, request, timer routine, response routine). With this routine, a "Smart" Alarm Clock would be easy to create.

2) An Appletalk socket completion routine that automatically adds a request whenever someone sends a packet to that computer. If this socket is loaded into low memory, it would create a simple pager utility.

3) A routine that would install an Icon Notification request, then set up a timer so that the request (and icon) would be automatically removed a few moments later.

Listing: HeyYou.pas

```
( HeyYou program for MacTutor by Steve Sheets)
( Demonstrates uses of the Notification Manager. Creates
some simple String Notifications, some Sound)
( Notifications, some Small Icon Notifications (immediately
and after a delay).)
```

```
program HeyYou;
( Resource Constants. )
const
    SICNdiamond = 0;
    SICNheyyou = 500;
    SNDbeep = 1;
    SNDclick = 2;
    SNDbong = 3;
    SNDmonkey = 4;
    ALERTabout = 500;

( Notification Manager Queue Record. )
type
    NMRec = record
        qLink: QElemPtr;
        qType: INTEGER;
        nmFlags: INTEGER;
        nmPrivate: LONGINT;
        nmReserved: INTEGER;
        nmMark: INTEGER;
        nmSIcon: Handle;
        nmSound: Handle;
        nmStr: StringPtr;
        nmResp: ProcPtr;
        nmRefCon: LONGINT;
    end;
    NMPtr = ^NMRec;

( Program Global Variables. Menus, Notification Requests,
Strings and Flags. )
var
    myM1, myM2, myM3: MenuHandle;
    StrRec, IconRec: NMRec;
    StrStr, IconStr: Str255;
    doneFlag, IconFlag: BOOLEAN;
    Timer: LONGINT;
```

```
( Notification Manager Glue routines. )
( Install Notification Request into Queue.)
function NMInstall (nmReqPtr: QElemPtr): OSErr;
inline
    $205F, $A05E, $3E80;
( MOVE.L (SP)+,A0 )
( _NMInstall )
( MOVE.W D0,(SP) )
```

```
( Remove Notification Request into Queue. )
function NMRemove (nmReqPtr: QElemPtr): OSErr;
inline
    $205F, $A05F, $3E80;
```

```
( MOVE.L (SP)+,A0 )
( _NMRemove )
( MOVE.W D0,(SP) )
```

```
(Checks to see if this is System 6.0 or above)
function CheckSystem: Boolean;
var
    theWorld: SysEnvRec;
begin
    if SysEnvirons(1, theWorld) = noErr then
        CheckSystem := (theWorld.systemVersion >= $0600)
    else
        CheckSystem := FALSE;
end;
```

```
( Display About Info. )
procedure DoAbout;
var
    dummy: INTEGER;
begin
    dummy := Alert(ALERTabout, nil);
end;
```

```
( First Test of Notification Manager. )
( Displays a simple String. The Notification Request is
automatically removed after it is completed. )
procedure NotifStr (St: Str255);
var
    E: OSErr;
begin
    with StrRec do
        begin
            qType := 8;
( No Mark, Icons or Sounds. )
            nmMark := 0;
            nmSIcon := nil;
            nmSound := nil;
( This String. )
            StrStr := St;
            nmStr := @StrStr;
( Automatically Removed when Completed. )
            nmResp := POINTER(-1);
            nmRefCon := 0;
        end;
        E := NMInstall(@StrRec);
    end;
```

```
( Call by Notification Manager after a Sound Notification has
been completed. Removes )
( the entry from Notification queue, releases SND resource
if there is one, disposes of )
( string handle if there is one and disposes of the request
record. )
```

```
procedure MySndResponse (nmReqPtr: QElemPtr);
var
    aPtr: NMPtr;
    E: OSErr;
begin
    aPtr := NMPtr(nmReqPtr);
    E := NMRemove(nmReqPtr);
    if (aPtr^.nmSound <> nil) and (aPtr^.nmSound <> POINTER(-1)) then
        ReleaseResource(aPtr^.nmSound);
    if aPtr^.nmRefCon <> 0 then
        begin
            HUnlock(Handle(aPtr^.nmRefCon));
            DisposalHandle(Handle(aPtr^.nmRefCon));
        end;
    DisposPtr(Ptr(nmReqPtr));
end;
```

```
( Second Test of Notification Manager. )
( Displays a String and plays a sound. Allocates memory
for the request record and the string.)
```

```

( If Sound is -1, the System Beep is used. Any other
number, indicates a SND resource, and)
( that resource is loaded in. MySndResponse is setup so
that it will be called after Notification is completed.)
procedure NotifSnd (St: Str255; Sn: INTEGER);
var
  E: OSerr;
  tempRec: NMPtr;
  StrHdl: StringHandle;
begin
  tempRec := NMPtr(NewPtr(SIZEOF(NMRec)));
  with tempRec do
  begin
    qType := 8;
    ( No Marks or Icons. )
    nmMark := 0;
    nmSIcon := nil;
    ( System Beep or SND resource. )
    if (Sn = -1) then
      nmSound := POINTER(-1)
    else
      nmSound := GetResource('snd', Sn);
    ( If String, allocate memory for it. )
    if St = '' then
      begin
        nmStr := nil;
        nmRefCon := 0;
      end
    else
      begin
        StrHdl := NewString(St);
        HLock(Handle(StrHdl));
        nmRefCon := ORD4(StrHdl);
        nmStr := StrHdl;
      end;
  end;
  ( Call MySndResponse to remove resource when completed. )
  nmResp := @MySndResponse;
end;
E := NMInstall(QElemPtr(tempRec));
end;

( Third Test of Notification Manager. )
( Displays a String, displays a Small Icon and Marks the
Current Application. )
( This Notification is not removed until the User selects
"Remove Icon" )
( from the Menu or the Application is Quit. )
procedure NotifIcon (St: Str255; IC: INTEGER);
var
  E: OSerr;
begin
  if not IconFlag then
  begin
    IconFlag := TRUE;
    with IconRec do
    begin
      qType := 8;
      ( Current Application is Marked (in MultiFinder). )
      nmMark := 1;
      ( Small Icon is used. )
      nmSIcon := GetResource('SICN', IC);
      ( No SND. )
      nmSound := nil;
      ( This String. )
      ( Use String (if any). )
      if St = '' then
        nmStr := nil
      else
        begin
          IconStr := St;
          nmStr := @IconStr;
        end;
    end;
    ( No Completion Routine. )
    nmResp := nil;
  end;

```

```

    nmRefCon := 0;
  end;
  E := NMInstall(@IconRec);
end;
end;

( If an small Icon is still flashing, Removes it and releases
SND resource. )
procedure RemoveNotifIcon;
var
  E: OSerr;
begin
  if IconFlag then
  begin
    IconFlag := FALSE;
    E := NMRemove(@IconRec);
    if IconRec.nmSIcon <> nil then
      ReleaseResource(IconRec.nmSIcon);
    end;
  end;
end;

( Normal Mac Setup Procedure )
procedure SetUp;
var
  S: Str255;
begin
  doneFlag := FALSE;
  IconFlag := FALSE;
  Timer := 0;
  S := '1';
  S[1] := CHR(appleMark);
  myM1 := NewMenu(101, S);
  AppendMenu(myM1, 'NotifTest Source;(-)');
  AddResMenu(myM1, 'DRVVR');
  InsertMenu(myM1, 0);
  myM2 := NewMenu(102, 'File');
  AppendMenu(myM2, 'Quit');
  InsertMenu(myM2, 0);
  myM3 := NewMenu(103, 'Simple');
  AppendMenu(myM3, 'String Test #1;String Test #2;(-);Bong
Sound Test;Click Sound Test;Bong and Click Sound Test');
  AppendMenu(myM3, 'Monkey Sound Alone Test - no
alert;System Error Sound Test;(-);Diamond Icon Test');
  AppendMenu(myM3, 'HeyYou Icon Alone Test - no
alert;Delayed Icon Test;Remove Icon');
  InsertMenu(myM3, 0);
  DrawMenuBar;
end;

( Normal Mac Menu Command Routine. )
procedure DoCommand (mResult: LONGINT);
var
  theItem: INTEGER;
  theMenu: INTEGER;
  tempStr: Str255;
  tempInteger: INTEGER;
  tempLong: LONGINT;
begin
  theItem := LoWord(mResult);
  theMenu := HiWord(mResult);
  case theMenu of
    101:
      if theItem = 1 then
        DoAbout
      else
        begin
          GetItem(myM1, theItem, tempStr);
          tempInteger := OpenDeskAcc(tempStr);
        end;
    102:
      if theItem = 1 then
        doneFlag := TRUE;
    103:
      case theItem of

```

```

1:
  NotifStr('This is a String Test of the Notifica-
tion Manager. ');
2:
  begin
    GetDateTime(tempLong);
    IUTimeString(tempLong, TRUE, tempStr);
    tempStr := CONCAT('This is another String Test
of the Notification Manager. The Time is ', tempStr);
    NotifStr(tempStr);
  end;

4:
  NotifSnd('This is a Sound Test of the Notifica-
tion Manager using the Bong sound.', SNDbong);
5:
  NotifSnd('This is a Sound Test of the Notifica-
tion Manager using the Click sound.', SNDclick);
6:
  begin
    NotifSnd('This is a Sound Test of the Notifica-
tion Manager using the Bong sound.', SNDbong);
    NotifSnd('This is a Sound Test of the Notifica-
tion Manager using the Click sound.', SNDclick);
  end;
7:
  NotifSnd('', SNDmonkey);
8:
  NotifSnd('This is a Sound Test of the Notifica-
tion Manager using the System Error sound.', -1);

10:
  begin
    RemoveNotifIcon;
    NotifIcon('This is a Icon Test of the Notifica-
tion Manager using the Diamond Icon.', SICNdiamond);
  end;
11:
  begin
    RemoveNotifIcon;
    NotifIcon('', SICNheyYou);
  end;
( Call NotifIcon after 60 seconds (3600 ticks).)
12:
  Timer := TickCount + 3600;
13:
  RemoveNotifIcon;
  otherwise
  end;
  otherwise
  end;
  HiliteMenu(0);
end;

( Simple Mac Main Event Loop. Check to see if it is time to
display HeyYou Icon.)
(If Resuming under MultiFinder, Remove Icon (if any).)
procedure MainLoop;
const
  suspendResumeMessage = 1;
var
  myEvent: EventRecord;
  whichWindow: WindowPtr;
  tempStr: Str255;
  tempLong: LONGINT;
begin
  repeat
    if Timer <> 0 then
      if TickCount > Timer then
        begin
          RemoveNotifIcon;
          Timer := 0;
          GetDateTime(tempLong);
          IUTimeString(tempLong, TRUE, tempStr);

```

```

    NotifIcon(CONCAT('Hey You! The Time is ',
tempStr), SICNheyYou);
  end;

  if WaitNextEvent(everyEvent, myEvent, 0, nil) then
    case myEvent.what of
      mouseDown:
        case FindWindow(myEvent.where, whichWindow) of
          inSysWindow:
            SystemClick(myEvent, whichWindow);
          inMenuBar:
            DoCommand(MenuSelect(myEvent.where));
          otherwise
            end;
        App4Evt:
          if BitShift(myEvent.message, -24) = SuspendRe-
sumeMessage then
            if Odd(myEvent.message) then
              RemoveNotifIcon;
            otherwise
              end;
          until doneFlag;
        end;

( Deletes Menus and removes Icon Notification Request (if
any). )

```

```

procedure CloseDown;
begin
  RemoveNotifIcon;
  DeleteMenu(101);
  DeleteMenu(102);
  DeleteMenu(103);
  DisposeMenu(myM1);
  DisposeMenu(myM2);
  DisposeMenu(myM3);
end;

( Main Program . )
begin
  InitGraf(@thePort);
  InitFonts;
  FlushEvents(everyEvent, 0);
  InitWindows;
  InitMenus;
  TEInit;
  InitDialogs(nil);
  InitCursor;
  DoAbout;
  if CheckSystem then
    begin
      SetUp;
      MainLoop;
      CloseDown;
    end;
end.

```

Listing: HeyYou.r

```
/*_____
```

HeyYou.r - Resources for HeyYou
Notification Manager Example Program for MacTutor by Steve
Sheets

```
_____*/
```

```

#include "Types.r"

type 'HEY!' as 'STR '

resource 'HEY!' (0) (
  "Notification Manager Example for MacTutor by Steve Sheets"
);

```

```

resource 'BNDL' (128) (
    'HEY!',
    0,
    ( /* array TypeArray: 2 elements */
        /* [1] */
        'ICN#',
        ( /* array IDArray: 1 elements */
            /* [1] */
            0, 128
        ),
        /* [2] */
        'FREF',
        ( /* array IDArray: 1 elements */
            /* [1] */
            0, 128
        )
    )
);

resource 'FREF' (128) (
    'APPL',
    0,
    ""
);

resource 'ICN#' (128) (
    ( /* array: 2 elements */
        /* [1] */
        "$FFFF FFFF 807F FFFF 807F FFFF 807F FFFF"
        "$807F FFFF 807F C0FF 887F 003F 887E 001F"
        "$887C 000F 8078 0007 8079 5D47 8071 5143"
        "$8071 D9C3 8071 5083 8071 5C83 8070 0003"
        "$8070 0003 8070 0003 8071 5D43 8071 5543"
        "$87F1 D543 81F0 9543 81F0 9DC7 81F0 0007"
        "$81F0 000F 81E0 001F 8F80 007F 81FF FFFF"
        "$81FF FFFF 81FF FFFF 81FF FFFF FFFF FFFF",
        /* [2] */
        "$FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
        "$FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
        "$FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
        "$FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
        "$FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
        "$FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
        "$FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
    )
);

resource 'SICN' (500, purgeable) (
    ( /* array: 1 elements */
        /* [1] */
        "$7FFC 8002 ABAA AA2A BB3A AA12 AB92 8002"
        "$ABAA AAAA BAAA 92AA 93BA 8002 7FFC"
    )
);

```

```

);

resource 'ICON' (500) (
    "$FFFF FFFF 807F FFFF 807F FFFF 807F FFFF"
    "$807F FFFF 807F C0FF 887F 003F 887E 001F"
    "$887C 000F 8078 0007 8079 5D47 8071 5143"
    "$8071 D9C3 8071 5083 8071 5C83 8070 0003"
    "$8070 0003 8070 0003 8071 5D43 8071 5543"
    "$87F1 D543 81F0 9543 81F0 9DC7 81F0 0007"
    "$81F0 000F 81E0 001F 8F80 007F 81FF FFFF"
    "$81FF FFFF 81FF FFFF 81FF FFFF FFFF FFFF"
);

data 'SIZE' (-1) (
    "$5800 0006 4000 0006 4000"
);

resource 'ALRT' (500) (
    (40, 76, 164, 436),
    500,
    ( /* array: 4 elements */
        /* [1] */
        OK, visible, sound1,
        /* [2] */
        OK, visible, sound1,
        /* [3] */
        OK, visible, sound1,
        /* [4] */
        OK, visible, sound1
    )
);

resource 'DITL' (500) (
    ( /* array DITLarray: 3 elements */
        /* [1] */
        (84, 150, 104, 210),
        Button (
            enabled,
            "Ok"
        ),
        /* [2] */
        (20, 84, 74, 344),
        StaticText (
            disabled,
            "A Notification Manager example program f"
            "or MacTutor by Steve Sheets Requires Sys"
            "tem 6.0 or higher"
        ),
        /* [3] */
        (10, 10, 74, 74),
        Icon (
            disabled,
            500
        )
    )
);

```



MIDI Connections

MIDI User Interface



Kirk Austin
San Rafael, CA
Volume 5, Number 5

MIDI and the User Interface

This is an article that takes a look at the Macintosh's Dialog Box routines, and how they can be used effectively. The PopUp menus and ArrowEditText controls described here are an attempt at getting the greatest use out of limited space in a Dialog Box. Techniques like this are important in programs that make use of MIDI like sequencers and patch librarians, for example. Extensive use of dialogs can be found in these type of programs.

The significance of the user interface

This article is going to take a little bit of a different slant from the previous MIDI articles because I think it is important to talk about the human interface considerations for a bit. It seems that the thing that separates good Macintosh programs from average ones is the way that the user interface is implemented. A program has to look and feel like a Macintosh program should. The best way to accomplish this is to try to follow the Apple guidelines for doing things, and only using alternate techniques when absolutely nothing else will work. Probably the first book that aspiring Macintosh programmers should read is Human Interfaced Guidelines published by Addison Wesley.

Dialogs In MIDI programs

Most of the things that take place in a MIDI program happen in Dialog Boxes. These structures have been made very simple to use by Apple, and they can lead the user through a great deal of setup information that would otherwise be difficult to deal with. Getting the most out of a particular dialog box can be a challenge, though. There is limited space and a lot of things to be accomplished, so what is the best way to communicate with the user while making the most efficient use of space? In addressing this problem, I've found the pop-up menu idea to be very useful. It takes up very little space in a dialog box until it is selected. At that point it can display a large range of choices. Now, this can also be done with the list manager by displaying a scrolling list of choices, but that would take up more room than the pop-up menu solution. Another technique that I have made use of is what I call an Edit Text Arrow Control. I use this for the input of numbers. It can be used like a standard Edit Text Item, but, in addition, there is an up/down arrow control associated with the item that allows the user to select a number with the mouse instead of having to use the keyboard. In order to provide an example of these two techniques I have written a sample application that lets the user change the MIDI program number on a synthesizer. We'll get to the application in a short time, but first we should talk about the Pop-up Menus and the Arrow Edit Text Control.

t MIDI

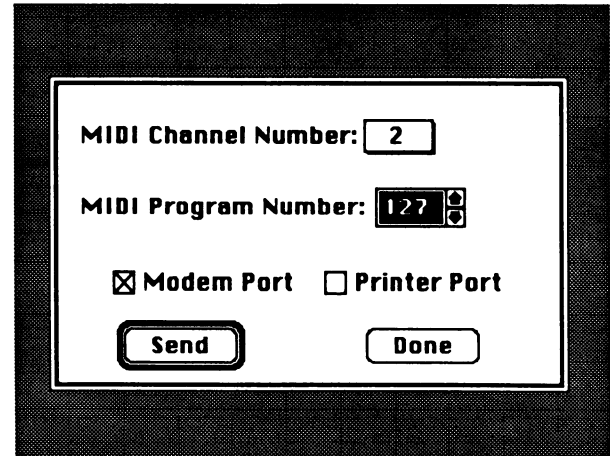


**Figure 1. Example Dialogbox
The RefCon field**

Fortunately for us, Apple gave us the RefCon field of a window record. This is a longint that we can use however we want. I use this field to give each Dialog a unique number that can be used to calculate a number for the User Items in a Dialog. Both the Pop-Up Menus and the ArrowEditTextCtl use this number to identify themselves.

PopUp Menus

These routines are based on some samples distributed by Apple, and produce the standard Pop-Up Menus as described in the User Interface Guidelines. The way I have implemented them is closely related with the items in the dialog box that I created with the Resource Editor. This is because the PopUp menu is made up of two dialog items, a static text item and a user item. My routines expect the UserItem item Number to be one greater than the staticText item number. This works out to be a logical way to create the PopUp menu location from the Resource Editor. Just create the StaticText first, then create the UserItem. In addition to the items in the dialog box you have to define the menu as a resource with the Resource Editor. The important thing here is that the menu resource number has to be the UserItem item number plus the value in the Dialog's RefCon field (whew). I set the dialog's RefCon field to 200 in my program, and if I were going to use additional dialog's I could assign RefCon values of 300, 400, 500, etc. to them. If these numbering relationships are kept straight then it is pretty easy to create and use PopUp Menus. There are only two routines to use for all of the PopUp Menus in your application:

```

procedure DrawPopUp(TheDialog : DialogPtr; theItem : integer);
function PUMenuAction(theDialog : DialogPtr; PopUpItem : integer)
: integer;

```

The DrawPopUp procedure is called by PUMenuAction and also by the Dialog Manager after you install it as your UserItem drawing proc. Otherwise, you never actually call this routine yourself. PUMenuAction is what you call when there's a mousedown in the UserItem that is associated with the particular PopUp menu. PUMenuAction returns the menu item number as its result.

The ArrowEditTextCtl

EditText items in dialog boxes are great! They are pretty easy to use, and provide a standard way for the user to enter information into the Mac. The only thing about them that bothers me sometimes is that they require the user to type in the information, thereby taking the hand off of the mouse. I wanted a way to change text in an EditText item that would work from the mouse, so I came up with the ArrowEditTextCtl. The ArrowEditTextCtl is a regular EditText item with a frame drawn around it. Along the right side of the frame are the up and down arrows that increment or decrement the number in the EditText item. The ArrowEditTextCtl is a structure that also requires two dialog items defined for it. There is an EditText box and a UserItem. The EditText item number must be one less than the UserItem item number. Some care must be taken when creating the EditText and UserItem's. First create your EditText box (which must be 16 pixels high), then create the UserItem to be the same coordinates except -3 top, -3 left, +3 bottom, and +13 right. Once you do that the control will draw itself properly. There are two routines that are used to implement the ArrowEditTextCtl's in a program:

```

procedure DrawArrowETCtl(TheDialog : DialogPtr; theItem :
integer);
function ArrowCtlAction(TheDialog : DialogPtr; theItem :
integer; limitLo : integer; limitHi : integer) : integer;

```

All you have to do with the DrawArrowETCtl procedure is install it as the UserItem's drawing proc. Then you call ArrowCtlAction in response to a mousedown in the UserItem. The way it is written this routine will return the integer that is selected by either using the up and down arrows, or typing it in the EditText box.

Using the MIDI library

The MIDI library is a set of routines that were described in the December issue of MacTutor, so I don't think we need to spend any more time on them here. All the program is doing as far as MIDI is concerned is sending a program change message, which is pretty trivial, really. [There are some updates to Kirk's MIDI library from the last published version. They are not printed in the article, but they are included on the source code disk for this issue. -ed]

The MIDI Ctl's Application

The application is pretty simple, really. It just gives you a menu with one item in it. When you select the menu item a dialog

box appears on the screen. This is the dialog that demonstrates the use of the PopUp menu and the ArrowEditTextCtl. The dialog box is created as 'not visible' by the Resource Editor. This lets us do our GetNewDialog call and install all of the drawing procedures for the UserItem's as well as set up the RefCon field of the dialog. After all of that is done we do a call to ShowWindow to make the dialog visible. This works out to be much faster than watching the dialog draw all of its items on the screen. The function MyDialogFilter is used to watch for the mouse clicks in our UserItem's as well as check for keyboard input and making sure that a valid number gets typed into the EditText item. The rest of the program is your standard vanilla Macintosh application.

Listing: MIDICtl's

(Kirk Austin, 4/9/88)

(This shows how to use a dialog box in a MIDI program)

PROGRAM ShellExample;

USES

ROM85, ColorQuickDraw, ColorMenuMgr, LSPMIDI;

(Global Constants)

CONST

Null = '';

AppleMenuID = 1;

FileMenuID = 2;

EditMenuID = 3;

MIDIMenuID = 4;

PopMenuID = 206; (Item number plus the value in the Dialog's RefCon)

AboutID = 200;

MIDIIDialog = 201;

MIDIIDialogRefCon = 200;

(Items in our dialog box)

OKOutline = 8;

AEditText = 3;

ArrowETCtl = 4;

iPopPrompt = 5; (the Prompt staticText)

iPopUp = 6; (the Pop-up userItem)

ModemCheckBox = 9;

PrinterCheckBox = 10;

(Global Types)

TYPE

MIDIPrgData = RECORD

MIDIChan : integer;

MIDIProg : integer;

ModemActive : boolean;

PrinterActive : boolean;

END;

MPDPtr = ^MIDIPrgData;

MPDHdle = ^MPDPtr;

(Global Variables)

VAR

myMenus : ARRAY[AppleMenuID..MIDIMenuID] OF MenuHandle;

Done : Boolean; (true when user selects quit)

TextCursor : CursHandle; (handle to the text entry cursor)

ClockCursor : CursHandle; (handle to the waiting watch

cursor)

PopMenuHdle : MenuHandle;

TheMPDPtr : MPDPtr;

TheMPDHdle : MPDHdle;

TheResHdle : handle;

TheResRefNum : integer;

```

TheType : integer;
TheHandle : Handle;
TheRect : Rect;

PROCEDURE ShowAbout;
VAR
    theDlog : DialogPtr;
    oldPort : GrafPtr;
BEGIN
    GetPort(oldPort);
    theDlog := GetNewDialog(AboutID, NIL, Pointer(-1));
    SetPort(theDlog);
    DrawDialog(theDlog);
    WHILE NOT Button DO
        SystemTask;
    DisposDialog(theDlog);
    SetPort(oldPort);
END;

PROCEDURE LaunchIt (mode : integer;
                    VAR fName : Str255);
(The compiler has just pushed a word for the mode, and a
pointer to the string)
INLINE
    $204F, (movea.l a7, a0; (a0) is ptr to string, 4(a0) is mode)
    $A9F2; {Launch}

PROCEDURE DoXfer;
VAR
    where : Point;
    reply : SFReply;
    vRef : integer;
    thefName : Str255;
    textType : SFTypelist;
BEGIN
    where.h := 80;
    where.v := 55;
    textType[0] := 'APPL';
    SFGetFile(where, Nil, NIL, 1, textType, NIL, reply);
    WITH reply DO
        IF NOT good THEN
            thefName := Nil
        ELSE
            BEGIN
                thefName := fName;
                vRef := vRefNum
            END;
        IF thefName <> Nil THEN
            BEGIN
                Done := true;
                IF SetVol(NIL, vRef) = noErr THEN
                    LaunchIt(0, thefName)
                END
            END
        END;
END;

PROCEDURE DrawOKOutline (theDialog : DialogPtr;
                        theItem : INTEGER);
VAR
    savePen : PenState;
BEGIN
    GetPenState(savePen); {save the old pen state}
    GetDItem(TheDialog, theItem, TheType, TheHandle, TheRect);
(get the item's rect)
    PenSize(3, 3); {make the pen fatter}
    InsetRect(TheRect, -4, -4);
    FrameRoundRect(TheRect, 16, 16); {draw the ring}
    SetPenState(savePen); {restore the pen state}
END; {DrawOKOutline}

(DrawPopUp procedure was made to be as general as possible)
(The main thing to remember is that it expects PopUpMenuID to)
(be theItem + TheRefCon of the Dialog.Also, Prompt item number)
(must be 1 less than the Pop Up Menu item number)
PROCEDURE DrawPopUp (TheDialog : DialogPtr;

```

```

                        TheItem : integer);
CONST
    (constants for positioning the default item within its box)
    leftSlop = 13; {leave this much space on left of title}
    rightSlop = 5; { this much on right}
    botSlop = 5; { this much below baseline}
VAR
    TheType : integer;
    TheHandle : handle;
    r : Rect;
    TheString : Str255;
    newWid, newLen, wid : INTEGER;
    TheMenuHdle : MenuHandle;
    TheMenuItem : integer;
    i : integer;
    TheChar : char;
    TheRefCon : integer;
    MenuItemsCount : integer;
BEGIN
    (Get the menu that is associated with this Dialog Item
    (TheItem + TheRefCon))
    TheRefCon := LoWord(GetWRefCon(WindowPtr(TheDialog)));
    TheMenuHdle := MenuHandle(GetResource('MENU', TheItem +
    TheRefCon));
    (Now, figure out which menu item is the current selection by
    scanning for a check mark)
    MenuItemsCount := CountMItems(TheMenuHdle);
    TheMenuItem := 0;
    i := 1;
    REPEAT
        GetItemMark(TheMenuHdle, i, TheChar);
        IF TheChar = char(CheckMark) THEN
            TheMenuItem := i;
            i := i + 1;
        UNTIL (TheMenuItem <> 0) OR (i = MenuItemsCount + 1);
        IF TheMenuItem = 0 THEN
            BEGIN
                SetItemMark(TheMenuHdle, 1, CHR(checkMark));
            (check the first item)
            TheMenuItem := 1;
            END;
        GetItem(TheMenuHdle, TheMenuItem, TheString);
        (get currently-selected item)
        GetDItem(TheDialog, TheItem, TheType, TheHandle, r);
        (set up the rectangle)
        WITH r DO
            BEGIN
                InsetRect(r, -1, -1); {make it a little bigger}
                (Make sure title fits. Truncate it add an ellipses ("...")
                (if it doesn't (by the way, "... is option-semicolons)
                wid := (right - left) - (leftSlop + rightSlop);
                (available string area)
                newWid := StringWidth(TheString); {get current width}
                IF newWid > wid THEN
                    BEGIN {doesn't fit - truncate it}
                        newLen := LENGTH(TheString);
                        (current length in characters)
                        wid := wid - CharWidth('...');
                        (subtract width of ellipses)

                        REPEAT (until fits (or we run out of characters))
                            (drop the last character and its width)
                            newWid := newWid - CharWidth(TheString[newLen]);
                            newLen := PRED(newLen);
                        UNTIL (newWid <= wid) OR (LENGTH(TheString) = 0);
                        (add the ellipses character)
                        newLen := SUCC(newLen); {one more char}
                        TheString[newLen] := '...'; {it's the ellipses}
                        TheString[0] := CHR(newLen); {fix the length}
                    END;

                    (draw the box and its drop shadow)
                    FrameRect(r);
                    MoveTo(right, top + 2);
                END;
            END;
        END;
    END;

```

```

        LineTo(right, bottom);
        LineTo(left + 2, bottom);
    (draw the string)
        MoveTo(left + LeftSlop, bottom - BotSlop);
        DrawString(TheString);
    END;
END; (DrawPopUp)

FUNCTION PUMenuAction (TheDialog : DialogPtr;
    PopUpItem : integer) : integer;
VAR
    popLoc : Point;
    newChoice : INTEGER;
    chosen, ignoreLong : LongInt;
    TheString : Str255;
    TheItem : integer;
    TheType : integer;
    TheHandle : handle;
    PromptRect : rect;
    PopUpRect : rect;
    TheMenuHdle : MenuHandle;
    TheRefCon : integer;
    TheMenuID : integer;
    TheMenuItem : integer;
    i : integer;
    TheChar : char;
    MenuItemsCount : integer;
BEGIN
    PUMenuAction := 0;
    (Get the menu that is associated with this Dialog Item
    (PopUpItem + TheRefCon))
    TheRefCon := LowWord(GetWRefCon(WindowPtr(TheDialog)));
    TheMenuID := PopUpItem + TheRefCon;
    TheMenuHdle := MenuHandle(GetResource('MENU', PopUpItem +
    TheRefCon));
    (Now, figure out which menu item is the current selection by
    scanning for a check mark)
    MenuItemsCount := CountMItems(TheMenuHdle);
    TheMenuItem := 0;
    i := 1;
    REPEAT
        GetItemMark(TheMenuHdle, i, TheChar);
        IF TheChar = char(CheckMark) THEN
            TheMenuItem := i;
            i := i + 1;
        UNTIL (TheMenuItem <> 0) OR (i = MenuItemsCount + 1);
    (Call PopUpMenuSelect and let user drag around. Note that )
    ((top, left) parameters to PopUpMenuSelect are our item's, )
    (converted to global coordinates.)
    GetDItem(TheDialog, PopUpItem - 1, TheType, TheHandle,
    PromptRect);
    GetDItem(TheDialog, PopUpItem, TheType, TheHandle,
    PopUpRect);
    InvertRect(PromptRect); (highlight the prompt)
    InsertMenu(TheMenuHdle, -1); (insert our menu in menu list)
    PopLoc := PopUpRect.TopLeft; (copy our item's topleft)
    LocalToGlobal(PopLoc); (convert back to global coords)
    CalcMenuSize(TheMenuHdle); (Work around Menu Mgr bug)
    WITH popLoc DO
        chosen := PopUpMenuSelect(TheMenuHdle, v, h, TheMenu-
    Item);
    InvertRect(PromptRect); (unhighlight the prompt)
    DeleteMenu(TheMenuID); (remove our menu from menu list)
    (Was something chosen?)
    IF chosen <> 0 THEN
        BEGIN (yep, something was chosen)
            newChoice := LowWord(chosen); (get chosen item number)
            IF newChoice <> TheMenuItem THEN
                BEGIN
                    (the user chose an item other than the current one)
                    SetItemMark(TheMenuHdle, TheMenuItem, ' ');
                (unmark the old choice)
                    SetItemMark(TheMenuHdle, newChoice,
                    CHR(checkMark)); (mark the new choice)

```

```

        PUMenuAction := newChoice;
    (Draw the new title)
    EraseRect(PopUpRect);
    DrawPopUp(theDialog, iPopUp);
    END; (if this choice was not the current choice)
    END; (if something was chosen)
END; (of PUMenuAction)

(EditText number must be 1 less than ArrowUserItem number)
PROCEDURE DrawArrowETCt1 (TheDialog : DialogPtr;
    TheItem : integer);
VAR
    theType : Integer;    ( the type of dlog item )
    theHandle : Handle;    ( Handle to the item )
    theRect : Rect;    ( rect which encloses the item )
    Height : Integer;
    HalfHeight : integer;
    ArrowRect : rect;
BEGIN
    GetDItem(TheDialog, TheItem, theType, theHandle, theRect);
    (get handle to control)
    FrameRect(TheRect);
    InsetRect(TheRect, 2, 2);
    TheRect.right := TheRect.right - 10;
    FrameRect(TheRect);
    GetDItem(TheDialog, TheItem, theType, theHandle, theRect);
    (get handle to control)
    Height := TheRect.bottom - TheRect.top;
    HalfHeight := Height DIV 2;
    HalfHeight := TheRect.bottom - HalfHeight;
    TheRect.left := TheRect.right - 11;

    EraseRect(TheRect);
    FrameRect(TheRect);

    MoveTo(TheRect.left, HalfHeight); (draw bold center line)
    LineTo(TheRect.right - 1, HalfHeight);
    MoveTo(TheRect.left, HalfHeight - 1);
    LineTo(TheRect.right - 1, HalfHeight - 1);

    ArrowRect.top := TheRect.top + 4;    (draw up arrow)
    ArrowRect.bottom := HalfHeight - 2;
    ArrowRect.left := TheRect.left + 3;
    ArrowRect.right := TheRect.right - 3;
    FillRect(ArrowRect, black);
    MoveTo(ArrowRect.left - 1, ArrowRect.top + 1);
    LineTo(ArrowRect.right, ArrowRect.top + 1);
    MoveTo(ArrowRect.left + 1, ArrowRect.top - 1);
    LineTo(ArrowRect.right - 2, ArrowRect.top - 1);
    MoveTo(ArrowRect.left + 2, ArrowRect.top - 2);
    LineTo(ArrowRect.left + 2, ArrowRect.top - 2);

    ArrowRect.top := HalfHeight + 2;
    ArrowRect.bottom := TheRect.bottom - 4;
    ArrowRect.left := TheRect.left + 3;
    ArrowRect.right := TheRect.right - 3;
    FillRect(ArrowRect, black);
    MoveTo(ArrowRect.left - 1, ArrowRect.bottom - 2);
    LineTo(ArrowRect.right, ArrowRect.bottom - 2);
    MoveTo(ArrowRect.left + 1, ArrowRect.bottom);
    LineTo(ArrowRect.right - 2, ArrowRect.bottom);
    MoveTo(ArrowRect.left + 2, ArrowRect.bottom + 1);
    LineTo(ArrowRect.left + 2, ArrowRect.bottom + 1);
END;

FUNCTION ArrowCtl1Action (TheDialog : DialogPtr;
    TheItem : integer;
    LimitLo : integer;
    LimitHi : integer) : integer;
VAR
    theType : Integer;
    theHandle : Handle;
    theRect : Rect;

```



```

TheNum : longint;
TheString : Str255;
Height : Integer;
HalfHeight : integer;
ArrowRect : rect;
ThePoint : point;
Inverted : boolean;
HowLong : longint;
TickResult : longint;
UpArrowRect : rect;
DnArrowRect : rect;
TheTEHandle : handle;
GoingUp : boolean;

PROCEDURE BtnDelay (ticks : integer);
VAR
    dummy : longint;
    i : integer;
BEGIN
    i := ticks;
    IF ticks = 0 THEN
        i := 1;
    REPEAT
        Delay(1, dummy);
        i := i - 1;
    UNTIL (i = 0) OR NOT button;
END;

BEGIN
    GetDItem(TheDialog, TheItem - 1, theType, theHandle,
theRect); (get handle to control)
    TheTEHandle := TheHandle;
    GetIText(TheTEHandle, TheString);
    StringToNum(TheString, TheNum);
    ArrowCtlAction := loword(TheNum);
    GetDItem(TheDialog, TheItem, theType, theHandle, theRect);
(get handle to control)
    Height := TheRect.bottom - TheRect.top;
    HalfHeight := Height DIV 2;
    HalfHeight := TheRect.bottom - HalfHeight;
    TheRect.left := TheRect.right - 11;
    UpArrowRect := TheRect;
    UpArrowRect.bottom := HalfHeight;
    DnArrowRect := TheRect;
    DnArrowRect.top := HalfHeight;
    Inverted := false;
    HowLong := 22;
    GetMouse(ThePoint);
    IF (PtInRect(ThePoint, TheRect)) AND Stilldown THEN (we
need to hilite an arrow)
    BEGIN
        IF PtInRect(ThePoint, UpArrowRect) THEN
            BEGIN
                ArrowRect := UpArrowRect;
                GoingUp := true;
            END
        ELSE
            BEGIN
                ArrowRect := DnArrowRect;
                GoingUp := false;
            END;
        REPEAT
            GetMouse(ThePoint);
            IF NOT PtInRect(ThePoint, ArrowRect) THEN
                BEGIN
                    IF inverted THEN
                        BEGIN
                            TheRect := ArrowRect;
                            InsetRect(TheRect, 1, 1);
                            InvertRect(TheRect);
                            Inverted := false;
                        END;
                    END
                ELSE

```

```

BEGIN
    IF NOT Inverted THEN
        BEGIN
            TheRect := ArrowRect;
            InsetRect(TheRect, 1, 1);
            InvertRect(TheRect);
            Inverted := true;
        END;
    GetIText(TheTEHandle, TheString);
    StringToNum(TheString, TheNum);
    IF GoingUp THEN
        BEGIN
            IF TheNum <> LimitHi THEN
                BEGIN
                    TheNum := TheNum + 1;
                    IF TheNum > LimitHi THEN
                        TheNum := LimitHi;
                    ArrowCtlAction := loword(TheNum);
                    NumToString(TheNum, TheString);
                    SetIText(TheTEHandle, TheString);
                    SelIText(TheDialog, TheItem - 1, 0,
32767);
                END;
            END;
        ELSE
            BEGIN
                IF TheNum <> LimitLo THEN
                    BEGIN
                        TheNum := TheNum - 1;
                        IF TheNum < LimitLo THEN
                            TheNum := LimitLo;
                        ArrowCtlAction := loword(TheNum);
                        NumToString(TheNum, TheString);
                        SetIText(TheTEHandle, TheString);
                        SelIText(TheDialog, TheItem - 1, 0,
32767);
                    END;
                END;
            END;
            BtnDelay(HowLong);
            IF HowLong > 3 THEN
                HowLong := HowLong - 2;
            END;
            UNTIL NOT StillDown;
            DrawArrowETC1(TheDialog, TheItem);
        END;
    END; (ArrowCtlAction)

FUNCTION MyDialogFilter (theDialog : DialogPtr;
    VAR theEvent : EventRecord;
    VAR item : integer) : Boolean;
(function called by ModalDialog for every event that occurs)
( while in control. It is used to "filter" events so you)
( can do things when certain events occur. It is used to )
( change cursor to an I beam when editing text. The routine )
( also handles keyboard entry; limiting text input to numbers)
( and making return & enter the same as clicking OK button )

CONST
    CrCode = 13; (ASCII code for RETURN)
    EnterCode = 3; (ASCII ccode for ENTER)
    BsCode = 8; (ASCII code for Back Space)
    TabCode = 9; (ASCII ccode for Tab)
VAR
    mouseLocation : point; (holds coordinates of mouse loc.)
    TheHandle : Handle; (used for dummy purpose here)
    TheType : Integer; (used for dummy purpose here)
    TheRect : Rect; (used for dummy purpose here)
    TextBox1 : Rect; (defines area to test for cursor change)
    TheString : Str255;
    TheNum : longint;
    TheDlogRecPtr : DialogPeek;
    TheItem : integer;
    TheChoice : integer;
    TheMenuItem : integer;

```

```

i : integer;
TheChar : char;
TheMenuHdle : menuHandle;
ignoreLong : longint;
BEGIN
  MyDialogFilter := false; {let modalDialog handle event}
  GetDItem(theDialog, AEditText, TheType, TheHandle,
  TextBox1); {chg cur in area}

  CASE theEvent.what OF
    nullEvent : {nothing happening chk if cursor to change}
      BEGIN
        GetMouse(mouseLocation);
        IF PtInRect(mouseLocation, TextBox1) THEN
          SetCursor(TextCursor)
        ELSE
          SetCursor(arrow);
          GetIText(TheHandle, TheString); {see if someone
typed in an invalid number}
          StringToNum(TheString, TheNum);
          IF TheNum > 128 THEN
            BEGIN
              SysBeep(10);
              TheNum := 128;
              NumToString(TheNum, TheString);
              SetIText(TheHandle, TheString);
              SetIText(TheDialog, AEditText, 0, 32767);
(hilite the editable text)
            END;
            IF TheNum < 1 THEN
              BEGIN
                SysBeep(10);
                TheNum := 1;
                NumToString(TheNum, TheString);
                SetIText(TheHandle, TheString);
                SetIText(TheDialog, AEditText, 0, 32767);
(hilite the editable text)
              END;
            END;

            mouseDown :
              BEGIN {Click!}
                mouseLocation := theEvent.where;
(copy the mouse position)
                GlobalToLocal(mouseLocation); {convert to local
coordinates}

                {Was the click in a user item?}
                IF (FindDItem(theDialog, mouseLocation) + 1) = 4
THEN
                  BEGIN
                    TheMPDPtr^.MIDIProg :=
                    ArrowCtlAction(TheDialog, 4, 1, 128);
                    END; {if clicked in ArrowEditTextCtl userItem}

                    IF (FindDItem(theDialog, mouseLocation) + 1) =
iPopUp THEN
                      BEGIN {Clicked in the pop-up box}
                        TheChoice := PUMenuAction(TheDialog, iPopUp);
                        IF TheChoice <> 0 THEN
                          BEGIN
                            MyDialogFilter := TRUE; {dialog is over}
                            TheMPDPtr^.MIDIChan := TheChoice;
                            item := iPopUp; {have ModalDialog return
that the user changed items}
                          END;
                        END; {if clicked in our userItem}
                        END; {mousedown case}

                        keyDown, autokey : {to follow std. procedure, chk if
RETURN or ENTER was pressed}
                          BEGIN
                            IF (theEvent.message MOD 256) IN [crCode, enter-
Code] THEN

```

```

BEGIN
  GetDItem(theDialog, 1, TheType, TheHandle,
  TheRect);
  HiliteControl(ControlHandle(TheHandle), 1);
  {hilite the OK button}
  Delay(3, ignoreLong);
  HiliteControl(ControlHandle(TheHandle), 0);
  MyDialogFilter := true;
  Item := 1;
END
ELSE IF (theEvent.message MOD 256) IN [bsCode,
tabCode] THEN
  BEGIN
    END
    ELSE IF (Char(theEvent.message MOD 256) >= '0') AND
(Char(theEvent.message MOD 256) <= '9') THEN
      BEGIN
        TheDlogRecPtr := DialogPeek(theDialog);
        TheItem := TheDlogRecPtr^.editField + 1;
        { find out which EditText Item we are in }
        GetDItem(theDialog, TheItem, TheType,
TheHandle, TheRect);
        GetIText(TheHandle, TheString);
        IF (Length(TheString) > 2) THEN
          SetIText(TheHandle, Null);
          { set it to Null if there are more than 3 characters }
        END
      ELSE
        MyDialogFilter := true;
      END;
    OTHERWISE
      ;
    END; {of the CASE statment}
  END;

PROCEDURE SendMIDI;
VAR
  dummy : longint;
BEGIN
  IF TheMPDPtr^.ModemActive THEN
    BEGIN
      InitSCCA;
      TXMIDIA(TheMPDPtr^.MIDIChan + 1);
      TXMIDIA(TheMPDPtr^.MIDIProg - 1);
      Delay(1, dummy);
      ResetSCCA;
    END;
  IF TheMPDPtr^.PrinterActive THEN
    BEGIN
      InitSCCB;
      TXMIDIB(TheMPDPtr^.MIDIChan + 1);
      TXMIDIB(TheMPDPtr^.MIDIProg - 1);
      Delay(1, dummy);
      ResetSCCB;
    END;
  END; {of SendMIDI}

PROCEDURE ProcessMenu (codeWord : Longint); { menu selec}
VAR
  menuNum : Integer;
  itemNum : Integer;
  NameHolder : str255;
  dummy : Integer;
  yuck : boolean;
  oldPort : GrafPtr;
  aDialog : DialogPtr;
  ItemHit : integer;
  TheItemHandle : handle;
  TheItemType : integer; {type of the selected item}
  TheItemRect : rect; {bounding box of the selected item}
  TheNum : longint;
  TheString : Str255;
BEGIN
  IF codeWord <> 0 THEN { nothing was selected}

```

```

BEGIN
  menuNum := HiWord(codeWord);
  itemNum := LoWord(codeWord);
  CASE menuNum OF { the different menus}
    AppleMenuID :
      BEGIN
        IF itemNum < 3 THEN
          BEGIN
            ShowAbout;
          END
        ELSE
          BEGIN
            GetItem(myMenus[AppleMenuID], itemNum,
NameHolder);
            dummy := OpenDeskAcc(NameHolder);
          END;
        END;
      FileMenuID :
      BEGIN
        CASE ItemNum OF
          1 :
            BEGIN
              DoXfer;
            END;
          2 :
            BEGIN
              Done := true;
            END;
        END;
      END;
      EditMenuID :
      BEGIN
        yuck := SystemEdit(itemNum - 1);
      END;
      MIDIMenuID :
      BEGIN
        GetPort(oldPort);
        (Get a menu)
        PopMenuHdle := GetMenu(PopMenuID); (Create a
menu (its title is ignored))
        SetItemMark(PopMenuHdle, TheMPDPtr^.MIDIChan,
CHR(checkMark)); (check it)
        aDialog := GetNewDialog(MIDIDialog, NIL,
WindowPtr(-1));
        SetPort(aDialog);
        SetWRefCon(WindowPtr(aDialog), MIDIDialo-
gRefCon); (set the defaults)
        GetDItem(aDialog, AEditText, theType,
TheHandle, TheRect);
        TheNum := TheMPDPtr^.MIDIProg;
        NumToString(TheNum, TheString);
        SetIText(TheHandle, TheString);
        GetDItem(aDialog, ModemCheckBox, theType,
TheHandle, TheRect);
        IF TheMPDPtr^.ModemActive THEN
          SetCtlValue(ControlHandle(TheHandle), 1);
          GetDItem(aDialog, PrinterCheckBox, theType,
TheHandle, TheRect);
          IF TheMPDPtr^.PrinterActive THEN
            SetCtlValue(ControlHandle(TheHandle), 1);
            (Find out where our UserItems are, set their item handles to )
            (a pointer to the drawing procedures)
            GetDItem(aDialog, ArrowETCtl, theType,
TheHandle, TheRect);
            SetDItem(aDialog, ArrowETCtl, theType, @DrawAr-
rowETCtl, TheRect);
            GetDItem(aDialog, iPopUp, theType, TheHandle,
TheRect);
            SetDItem(aDialog, iPopUp, theType, @DrawPopUp,
TheRect);
            GetDItem(aDialog, OKOutline, theType,
TheHandle, TheRect);
            SetDItem(aDialog, OKOutline, theType, @DrawOK-
Outline, TheRect);

```

```

SellText(aDialog, 3, 0, 32767); (hilite editable
text)
ShowWindow(WindowPtr(aDialog)); (show window)
REPEAT
  ModalDialog(@MyDialogFilter, ItemHit);
  (will process all events while dialog is up)
  CASE itemHit OF
    1 :
      BEGIN (this is the Send Button item)
        SendMIDI;
      END;
    2 :
      BEGIN (this is the Done Button item)
        DisposDialog(aDialog);
        SetPort(ColdPort);
      END;
    iPopUp :
      BEGIN (this is the PopUpMenu item)
        SysBeep(1);
      END;
    ModemCheckBox :
      BEGIN (this is the Modem Check Box item)
        GetDItem(aDialog, ModemCheckBox,
theType, TheHandle, TheRect);
        IF TheMPDPtr^.ModemActive THEN
          BEGIN
            TheMPDPtr^.ModemActive := false;
          SetCtlValue(ControlHandle(TheHandle), 0);
          END
        ELSE
          BEGIN
            TheMPDPtr^.ModemActive := true;
          SetCtlValue(ControlHandle(TheHandle), 1);
          END;
        END;
    PrinterCheckBox :
      BEGIN (this is Printer Check Box item)
        GetDItem(aDialog, PrinterCheckBox,
theType, TheHandle, TheRect);
        IF TheMPDPtr^.PrinterActive THEN
          BEGIN
            TheMPDPtr^.PrinterActive := false;
          SetCtlValue(ControlHandle(TheHandle), 0);
          END
        ELSE
          BEGIN
            TheMPDPtr^.PrinterActive := true;
          SetCtlValue(ControlHandle(TheHandle), 1);
          END;
        END;
    OTHERWISE
      ;
    END; (case of item hit)
  UNTIL ItemHit = 2; (We're done with Dialog)
  END;
  END;
  HiliteMenu(0);
  END;
END;

PROCEDURE DealWithMouseDowns (theEvent : EventRecord);
VAR
  location : Integer;
  windowPointedTo : WindowPtr;
  mouseLoc : point;

```

```

windowLoc : integer;
VandH : Longint;
Height : Integer;
Width : Integer;
BEGIN
  mouseLoc := theEvent.where;
  windowLoc := FindWindow(mouseLoc, windowPointedTo);
  CASE windowLoc OF
    inMenuBar :
      BEGIN
        ProcessMenu(MenuSelect(mouseLoc));
      END;
    inSysWindow :
      BEGIN
        SystemClick(theEvent, windowPointedTo);
      END;
    inContent :
      BEGIN
        IF windowPointedTo <> FrontWindow THEN
          BEGIN
            SelectWindow(windowPointedTo);
          END;
        OTHERWISE
          BEGIN
            END;
          END;
        END;
      END;
  END;
END;

PROCEDURE DealWithKeyDowns (theEvent : EventRecord);
TYPE
  Trick = PACKED RECORD
    CASE boolean OF
      true : (
        long : Longint
      );
      false : (
        chr3, chr2, chr1, chr0 : char
      )
    END;
  END;
VAR
  CharCode : char;
  TrickVar : Trick;
BEGIN
  TrickVar.long := theEvent.message;
  CharCode := TrickVar.chr0;
  IF BitAnd(theEvent.modifiers, CmdKey) = CmdKey THEN (check
for a menu selection)
  BEGIN
    ProcessMenu(MenuKey(CharCode));
  END
END;

PROCEDURE DealWithActivates (theEvent : EventRecord);
VAR
  TargetWindow : WindowPtr;
BEGIN
  TargetWindow := WindowPtr(theEvent.message);
  IF Odd(theEvent.modifiers) THEN
    BEGIN
      SetPort(TargetWindow);
    END
  ELSE
    BEGIN
      END;
    END;
  END;
END;

PROCEDURE DealWithUpdates (theEvent : EventRecord);
VAR
  UpDateWindow : WindowPtr;
  tempPort : WindowPtr;
BEGIN
  UpDateWindow := WindowPtr(theEvent.message);

```

```

  GetPort(tempPort);
  SetPort(UpDateWindow);
  BeginUpDate(UpDateWindow);

  EndUpDate(UpDateWindow);
  SetPort(tempPort);
END;

PROCEDURE MainEventLoop;
VAR
  Event : EventRecord;
  ProcessIt : boolean;
  x : byte;
BEGIN
  REPEAT
    SystemTask;
    ProcessIt := GetNextEvent(everyEvent, Event); ( get the
next event in queue)
    IF ProcessIt THEN
      BEGIN
        CASE Event.what OF
          mouseDown :
            DealWithMouseDown(Event);
          AutoKey :
            DealWithKeyDowns(Event);
          KeyDown :
            DealWithKeyDowns(Event);
          ActivateEvt :
            DealWithActivates(Event);
          UpdateEvt :
            DealWithUpdates(Event);
          OTHERWISE
            BEGIN
              END;
            END;
          END;
        END;
      UNTIL Done;
    END;
  END;

PROCEDURE MakeMenus; ( get the menus & display them)
VAR
  index : Integer;
BEGIN
  FOR index := AppleMenuID TO MIDIMenuID DO
    BEGIN
      myMenus[index] := GetMenu(index);
      InsertMenu(myMenus[index], 0);
    END;
  AddResMenu(myMenus[AppleMenuID], 'DRVr');
  DrawMenuBar;
END;

( Program Starts Here )
BEGIN
  Done := false;
  FlushEvents(everyEvent, 0);
  ( initialize routines go here )
  (get the cursors we use and lock them down - no clutter)
  ClockCursor := GetCursor(watchCursor);
  TextCursor := GetCursor(iBeamCursor);
  HLock(Handle(ClockCursor));
  HLock(Handle(TextCursor));
  MakeMenus;
  TheResHdle := GetResource('MIDI', 128);
  HLock(TheResHdle);
  TheMPDHdle := MPDHdle(TheResHdle);
  TheMPDPtr := TheMPDHdle^;
  InitCursor;
  MainEventLoop;
  ChangedResource(TheResHdle);
  TheResRefNum := HomeResFile(TheResHdle);
  UpdateResFile(TheResRefNum);
  ReleaseResource(TheResHdle);
END.

```



Pascal Procedures

Modeless Search and Replace



Mark McBride
Oxford, CA
Volume 5, Number 5

Introduction

While working on development of a utility to speed integration of toolbox calls in MacFortran source code, I found myself looking for a search and replace routine. Perusal of *Macintosh Revealed* (Vol. 2) quickly led to the toolbox routine Munger (rhymes with plunger). A search of back issues of *MacTutor* also revealed the lack of a general discussion of Munger and doing find/replace (not even the generic multi-window text editor from the January 1987 issue). Thus, I decided to write a general purpose find/replace unit from a modeless dialog environment.

Basic Program Design

The find/replace unit and the accompanying example program are written in LS Pascal. Conversion to TML, Turbo, or MPW should be relatively straightforward. The find/replace unit is designed as a standalone unit that requires very little modification to the user's program to add its capabilities. The modifications required include changing the main event loop to handle modeless dialog events, adding a new menu, and inserting calls to the find/replace unit in your program. When calling the find/replace routines the user passes the current event and a handle for the editText to be searched. The program contains a simple text edit window, a modeless Find dialog, and a modeless Replace Dialog. The program supports desk accessories; cut, copy, and paste within the program (but not with desk accessories); and should be screen and machine size independent. The program requires a 512Ke or later machine. If run on an old 512k machine, the program alerts the user and exits back to the Finder.

Simple Text Editor

The demonstration program uses the ROM85 library enhancements for the textedit window which allowed me to write a barebones editor sufficient for demonstration purposes. The text window can be resized, zoomed, and automatically scrolled. However, there are no scroll bars, thus movement to text outside the visible region must be done with the arrow keys. Cut, copy, paste, and clear work properly. The text window contents cannot be loaded from disk, nor saved to disk. Upon startup (Figure 1), the text window contains Lincoln's Gettysburg Address retrieved from the resource fork.

Find/Replace Unit

The Find dialog allows the user to enter text characters and does an exact find. The search process can be initiated by clicking the 'Find' button or pressing the return key. The find routine works from the current selection point and does not wrap past the end of the textedit record. If found, the current selection is set to the string found and highlighted. If not found, the machine beeps. The find routine does not support wildcards nor

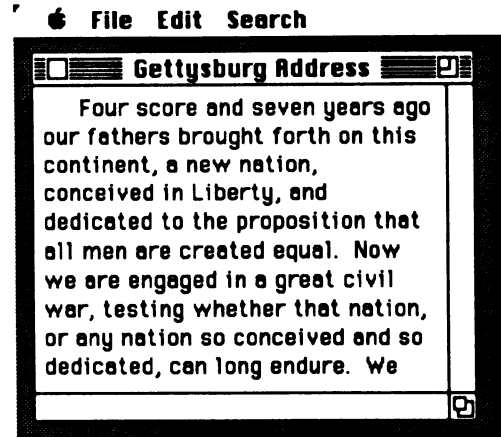


FIGURE 1

non-printing characters (e.g., control characters).

The Search dialog allows the user to enter both the search and replace strings. Initially, the 'Search' button is highlighted as the default button. Once the search string has been found (and selected, the search routine is just the find routine), the 'Replace' button is highlighted. If the 'Replace' button is selected then the occurrence of the search string previously found is replaced. The replace string can be null but the find string must contain at least one character. When the replace operation is successfully completed, the 'Search' button returns to the default button. Although I have not included a 'replace all' button (which could easily be integrated), the user can continually press the return key searching and replacing. Dialog cut, copy, and paste from the text edit scrap is supported within both the Find and Replace dialogs.

Handling Modeless Dialogs

To properly handle the modeless find and replace dialogs four major aspects of modeless dialog event handling must be observed carefully: main event loop changes, window events, dialog events, and command keys. Each element is relatively straightforward, but the combination of the four can lead to some confusion and unnecessary coding.

Main Event Loop

When a program contains modeless dialogs, the toolbox routine `IsDialogEvent` should be called after calling `GetNextEvent`. *Inside Mac* (page I-416) warns that the program should call `IsDialogEvent` even if `GetNextEvent` returns false. When `IsDialogEvent` returns true the program should drop into its routine to handle events for the modeless dialog. Thus, the program should call `GetNextEvent`, then `IsDialogEvent`, and

then start processing on the basis of the booleans returned by the two routines. If the program does a

```
if GetNextEvent(MyEvent) then
  begin
    if IsDialogEvent(MyEvent) then
      begin
        Do_Modeless(MyEvent);
```

... structure then null dialog events will not be passed to its modeless dialog routines. The alternative used by the program in the listing does the following:

```
EventFlag:=GetNextEvent(MyEvent);
if IsDialogEvent(MyEvent) then
  Do_Modeless(MyEvent)
else if EventFlag then
  begin
```

... According to *Inside Mac*, IsDialogEvent returns true for:

- activate or update events for a dialog window
- mouse-down events in the content region of an active dialog window
- any other event when a dialog window is active.

However, the last statement is misleading.

Window Events

The structure region of a window is composed of two parts: the content region and the window frame (see *Inside Mac*, pages I-271,272). In the standard document window, the window frame contains the drag, go-away, and zoom regions (if zoom was desired). These regions do not overlap with the content region in the standard document window, although they could in a custom window definition. Thus, if your modeless dialog uses a standard document window (or a noGrowDocument proc) then you must be careful in handling events. A mouse-down event in the window frame of a modeless dialog will return false for IsDialogEvent. The main event loop must handle mouse-downs in the drag, go-away, and zoom boxes of a modeless dialog when using standard window types. This does not pose a particular problem since handling multiple windows for growing, dragging, and zooming is a standard Mac programming task.

Dialog Handler

When IsDialogEvent returns true, the user should pass control to a routine that handles the modeless dialog event. This routine should call the function

```
DialogSelect(theEvent, whichDialog, ItemHit):boolean.
```

DialogSelect will return a result of true with the dialog pointer in whichDialog and the item number in ItemHit when there is

- a mouse-down, key-down, or auto-key event in an enabled editText item. DialogSelect will take appropriate action within the editText item.
- a mouse-down event, then mouse released within an enabled control item. DialogSelect will call TrackControl while the mouse button is down.
- a mouse-down event in any other enabled item.

DialogSelect returns a result of false with the dialog pointer, whichDialog, undefined and ItemHit undefined when there is

- an activate or update event for the dialog window. Dia-

logSelect will handle the redrawing of all items in the window (but will not handle redrawing of the default button).

- a mouse-down, key-down, or auto-key event in a disabled editText item.
- a key-down or auto-key event when no editText item exists.
- a mouse-down event in a control when the mouse is released outside the control.
- a mouse-down event in any other disabled item.

Thus, your modeless routine calls DialogSelect, letting the Dialog Manager handle actions while the user mouses around in the dialog. If the user indicates some action within an enabled dialog item, then the modeless routine takes appropriate action. If DialogSelect returns false then the modeless routine should do nothing. Two exceptions to handling returns of false from DialogSelect exist. First, if the modeless routine needs to take special actions for modeless update or activate events (e.g., redrawing the default button, showing selected text) then the routine must check for these types of events after DialogSelect returns false.

Second, DialogSelect does not check for the command key when handling key-down events. If some key-down events are special cases or if the command key equivalents for menu events (e.g., cut, copy, paste) are to be handled while the modeless dialog is the active window, then the modeless routine must check for key-down events *prior* to calling DialogSelect. The find/replace unit in the listing checks for edit menu command key equivalents, find/replace menu command key equivalents, and the return key as the default button.

Find/Replace Routines

Implementing a find/replace unit can take many alternative forms. *Inside Mac* does not specify user-interface guidelines for find/replace actions leading to many different solutions existing. The implementation given in the listing uses a single modeless dialog and follows a simple logic within three routines: Find, Search, and Replace. These routines provide the setup to access the toolbox routine Munger. Munger provides the core find and replace actions on the specified text. Switching between find and replace options alters the appearance and action of the modeless dialog.

Logic Used

When in the Find mode, the user can enter a text string (Figure 2). When the user clicks the find button (or presses the 'return' key) the editText record is searched forward from the current selection. If located, the find string is highlighted in the editText record. If the string is not in the current visible region, a call to TESelView scrolls the string into view (Figure 3). If the string is not found, then the machine beeps.

When in the replace mode, the user must enter a find string while the replace string is optional (Figure 4). When the user clicks the find button (or presses 'return'), starting at the current selection, the find routine tries to locate and highlight the find string. If found, the search routine changes the default button to 'Replace' (Figure 5). When the user clicks the replace button (or

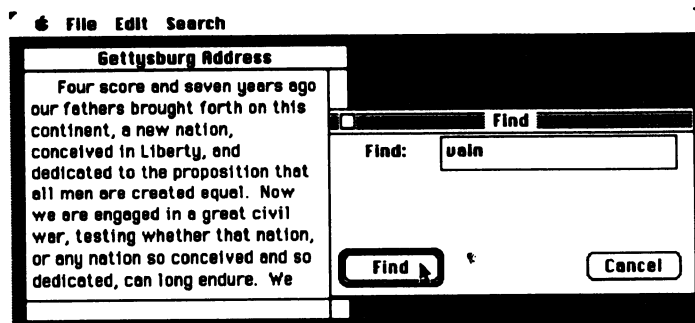


Figure 2

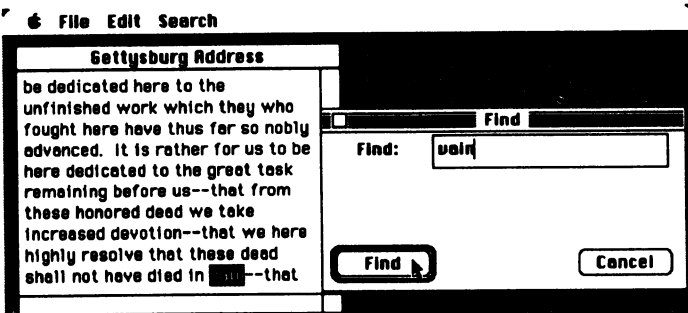


Figure 3

presses 'return') the find text is replaced with the replace text, leaving the current selection after the replacement text, and re-establishing the 'Search' button as the default button (Figure 6).

How It Works: Munger

The find, search, and replace routines serve to setup the proper information to send to and retrieve from the toolbox function Munger:

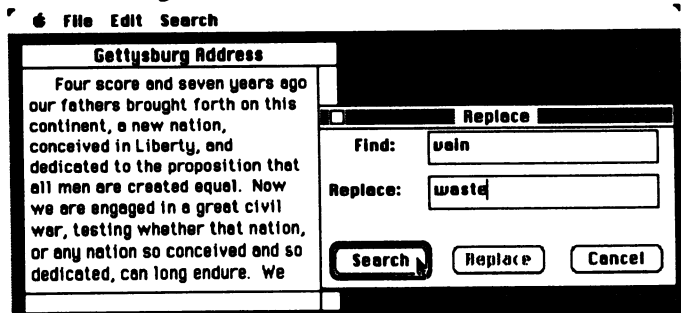


Figure 4

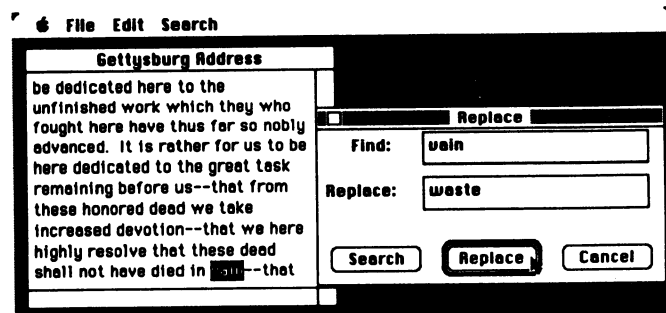


Figure 5

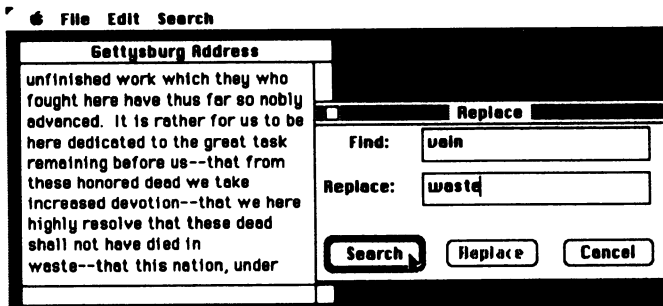


Figure 6

```
function Munger(h:handle; offset:longint; ptr1: ptr; len1:
longint;
```

```
ptr2: ptr; len2: longint): longint.
```

The handle, h, specifies a handle to destination text generated as a relocatable block allocated by the memory manager. Offset refers to the starting location of the operation in the destination text and must be less than the length of the destination text. Ptr1, len1 refers to the target text and ptr2, len2 refers to the replacement text. The function value returned is negative if the operation failed and positive if the specified operation was successful. The behavior of Munger depends on the values passed. The six possible cases are:

Case 1: ptr1<>nil, len1>0; ptr2<>nil, len2>0: the destination text is searched from the offset specified to the end of the text, replacing the first occurrence of the target string found with the replacement text. The function returns the offset of the first byte past the replaced text.

Case 2: ptr1=nil, len1>0; ptr2<>nil, len2>0: the destination text from the specified offset to offset + len1 is replaced by the replacement text. The function returns the offset of the first byte past the replaced text.

Case 3: ptr1=nil, len1<0; ptr2<>nil, len2>0: the destination text from the specified offset to the end of the destination text is replaced by the replacement text. The function returns the offset of the first byte past the replaced text.

Case 4: len1=0; ptr2<>nil, len2>0: the replacement text is inserted into the destination text at the specified offset. The function returns the offset of the first byte past the replaced text.

Case 5: ptr1<>nil, len1>0; ptr2=nil: the destination text is searched for the target string. Munger returns the offset at the beginning of the found string.

Case 6: ptr1<>nil, len1>0; ptr2<>nil, len2=0: the target string is deleted from the destination text instead of replaced. The function returns the offset at the byte where the text was deleted.

The find routine uses Case 5 to perform the search. The find routine retrieves the find string from the dialog, checking for a target string of length greater than zero. The replace pointer is set to nil. If the current selection is not an insertion point, the offset is set to the end of the current selection for the find dialog. The target length is retrieved and the replace length is set to zero. Finally, Munger is called. If the target string is found, the current selection is set to the target and scrolled into view.

The search routine locates the target string using the find

routine. If the find was successful, then the default button is set to the replace button and enabled. Once the target string is located, a user action to replace call the replace routine. The replace routine uses Case 1 for positive length replacement text and Case 6 for empty replacement text. The replace routine retrieves the target and replace text from the dialog edit fields, sets the offset to the current selection start of the destination text, and finally calls Munger. If Munger returns a positive value, the first occurrence of the target text was replaced. The editText is then recalibrated, the selection reset, and an invalrect call used to force an update of the editText window. Finally, the default button is set back to the Search button. If Munger returns a negative value, the replace failed and the machine beeps.

Concluding Remarks

The find/replace unit presented is simple in design, takes very little additional memory (approximately 5k), easily integrated into your application, and easily modified to suit your tastes. Most all the programming work is setting up the call to Munger and interpreting the results. The routine could also be changed to a modal dialog quickly. However, the modeless dialog environment seems more natural for a find/replace capability.

Listing: EnvironIntf.p

```
( File: EnvironsIntf.p)
( Copyright Apple Computer, Inc. 1987)
( All Rights Reserved)
```

```
UNIT EnvironsIntf;
```

INTERFACE

CONST

```
envMac = -1;      (returned by glue, if any)
envXL = -2;      (returned by glue, if any)
envMachUnknown = 0;
env512KE = 1;
envMacPlus = 2;
envSE = 3;
envMacII = 4;
envCPUUnknown = 0;      ( CPU types )
env68000 = 1;
env68010 = 2;
env68020 = 3;
envUnknownKbd = 0;      ( Keyboard types )
envMacKbd = 1;
envMacAndPad = 2;
envMacPlusKbd = 3;
envAExtendKbd = 4;
envStandADBKbd = 5;
( Errors )
envNotPresent = -5500; ( returned by glue. Official stuff)
envBadSel = -5501;      ( Selector non-positive )
envSelTooBig = -5502; ( Selector bigger than call can take)
```

TYPE

```
SysEnvRec = RECORD
    environsVersion : INTEGER;
    machineType : INTEGER;
    systemVersion : INTEGER;
    processor : INTEGER;
    hasFPU : BOOLEAN;
    hasColorQD : BOOLEAN;
    keyBoardType : INTEGER;
    atDrvVrsNum : INTEGER;
    sysVRefNum : INTEGER;
```

```
END;
```

```
FUNCTION SysEnvirons (versionRequested : INTEGER;
    VAR theWorld : SysEnvRec) : OSErr;
```

IMPLEMENTATION

```
FUNCTION SysEnvirons;
    External;
END.
```

Listing Globals.Pas:

```
UNIT SearchGlobs;
```

```
(File name:Globals.Pas)
(Function: Global Variables and Constants for Search Program)
(History: 7/02/88 MEM Initial Variables Added. )
( )
```

INTERFACE

USES

```
ROM85;
```

CONST

```
(window constants)
```

```
minWidth = 80;
minHeight = 80;
mBarHeightGlob = $BAA;
sBarWidth = 16;
TxtMargin = 4;
TxtWindID = 2;
AboutID = 20;
AlertID = 6;
I_OK = 1;
SpeechID = 100;
```

```
(Menu Resource ID's)
```

```
AppleID = 201;
FileID = 202;
EditID = 203;
SearchID = 204;
```

```
(Menu Choice Constants)
```

```
I_About = 1;
I_Quit = 1;
I_Undo = 1;
I_Cut = 3;
I_Copy = 4;
I_Paste = 5;
I_Clear = 6;
I_Find = 1;
I_Replace = 2;
I_Show_Text = 4;
```

```
(Global Variables)
```

VAR

```
(misc stuff)
```

```
doneFlag : boolean;
mBarHeight : integer;
```

```
(menu stuff)
```

```
AppleMenu : MenuHandle;
FileMenu : MenuHandle;
EditMenu : MenuHandle;
SearchMenu : MenuHandle;
```

```
(rectangles)
```

```
DragRect : Rect;
GrowRect : Rect;
ScreenRect : Rect;
TempRect : Rect;
```

```
(window stuff)
```

```
MyWindow : WindowPtr;
```

```
(text edit stuff)
```

```
hte : TEHandle;
```

```
(String stuff)
```

```
theString : Str255;
```

IMPLEMENTATION

```
END.
```


Listing: Init.Pas

```
UNIT Init;
{File name:Init.Pas}
{Function: Initialization Routines for Search Program}
{History: 7/02/88 MEM All routines Added. }
```

INTERFACE

```
    USES ROM85, EnvironsIntf, SearchGlobs;

    PROCEDURE InitMac;
    PROCEDURE CheckEnvirons;
    PROCEDURE InitTxtWind;
    PROCEDURE InitRects;
    PROCEDURE InitMenus;
```

IMPLEMENTATION

```
    PROCEDURE Crash;
    BEGIN
        ExitToShell;
    END;

    PROCEDURE InitMac;
    BEGIN
        MoreMasters;
        MoreMasters;
        MoreMasters;
        initCursor;
        FlushEvents(everyEvent, 0);
        doneFlag := false;
    END;

    PROCEDURE CheckEnvirons;
    VAR
        MyWorld : SysEnvRec;
        Version : integer;
        Err1, Err2 : OSErr;
    BEGIN
        Version := 1;
        Err1 := SysEnvirons(Version, MyWorld);
        IF (MyWorld.machineType < 1) THEN
            BEGIN
                ParamText('Program Requires Mac 512Ke or newer
machine', '', '', '');
                Err2 := NoteAlert(AlertID, NIL);
                ExitToShell;
            END;
        END;

    PROCEDURE InitRects;
    VAR
        MemPtr : ^Integer;
    BEGIN
        MemPtr := pointer(MBarHeightGlob);
        mBarHeight := MemPtr^;
        screenrect := ScreenBits.Bounds;
        SetRect(DragRect, ScreenRect.left + 4, ScreenRect.top +
mBarHeight + 4, ScreenRect.right - 4, ScreenRect.bottom - 4);
        SetRect(GrowRect, ScreenRect.left + minWidth,
ScreenRect.top + minHeight, ScreenRect.right - 8,
ScreenRect.bottom - 8);
    END;

    PROCEDURE InitTxtWind;
    VAR
        ItemHdl : Handle;
    BEGIN
        MyWindow := GetNewWindow(TxtWindID, NIL, Pointer(-1));
        SelectWindow(MyWindow);
        SetPort(MyWindow);
        WITH MyWindow^.portRect DO
            SetRect(TempRect, 0, 0, right - sBarWidth - 1, bottom -
sBarWidth - 1);
```

```
        InsetRect(TempRect, TxtMargin, TxtMargin);
        hte := TENew(TempRect, TempRect);
        HLock(Handle(hte));
        hte^.txFont := geneva;
        hte^.fontAscent := 12;
        hte^.lineHeight := 12 + 3 + 1; (ascent + descent +
leading)
        HUnlock(Handle(hte));
        ItemHdl := GetResource('TEXT', SpeechID);
        IF ItemHdl = NIL THEN
            BEGIN
                theString := 'Search and Replace Demo program. Use
find and replace to change text in this window.';
                TSEtText(Pointer(Ord4(@theString) + 1),
length(theString), hte);
            END
        ELSE
            BEGIN
                DetachResource(ItemHdl);
                SetHandleSize(hte^.hText, SizeResource(ItemHdl));
                hte^.hText := ItemHdl;
                hte^.teLength := SizeResource(ItemHdl);
                ReleaseResource(ItemHdl);
            END;
        TSEtSelect(0, 0, hte);
        TCElText(hte);
        TEAutoView(true, hte);
        TSElView(hte);
        TextFont(applFont);
    END;

    PROCEDURE InitMenus;
    VAR
        tempMenu : MenuHandle;
    BEGIN
        ClearMenuBar;
        tempMenu := GetMenu(AppleID);
        AddResMenu(tempMenu, 'DRVr');
        InsertMenu(tempMenu, 0);
        AppleMenu := tempMenu;
        tempMenu := GetMenu(FileID);
        InsertMenu(tempMenu, 0);
        FileMenu := tempMenu;
        tempMenu := GetMenu(EditID);
        InsertMenu(tempMenu, 0);
        EditMenu := tempMenu;
        tempMenu := GetMenu(SearchID);
        InsertMenu(tempMenu, 0);
        SearchMenu := tempMenu;
        DrawMenuBar;
        DisableItem(EditMenu, I_Undo);
    END;

END.
```

Listing: Utils.Pas

```
UNIT Utils;
{File name:Utils.Pas}
{Function: General Utility Routines for Search Program}
{History: 7/02/88 MEM Alert Error Routine Added. }
```

INTERFACE

```
    USES ROM85, SearchGlobs, Find_Dialog;

    PROCEDURE Do_About;
    PROCEDURE Do_Menu (theMenu, theItem : integer);
    PROCEDURE FrameDItem (dLog : DialogPtr; iNum : integer);
    PROCEDURE Update_Txt;
    PROCEDURE Act_Txt (theEvent : EventRecord);
    PROCEDURE FixText;
    PROCEDURE Grow_Txt (hSize, vSize : integer);
```

```

PROCEDURE Do_Quit;
IMPLEMENTATION
PROCEDURE Do_About;
VAR
    oldPort : GrafPtr;
    dlg : DialogPtr;
    itemHit : integer;
BEGIN
    GetPort(OldPort);
    dlg := GetNewDialog(AboutID, NIL, pointer(-1));
    FrameDialog(dlg, I_OK);
    ModalDialog(NIL, itemHit);
    DisposDialog(dlg);
    SetPort(OldPort);
END;

PROCEDURE Do_Menu;
VAR
    DaNum : integer;
    Bool : boolean;
    DAName : Str255;
    oldPort : GrafPtr;
BEGIN
    CASE theMenu OF
        AppleID :
            BEGIN
                CASE theItem OF
                    I_About :
                        BEGIN
                            Do_About;
                        END;
                    OTHERWISE
                        BEGIN
                            GetPort(oldPort);
                            GetItem(AppleMenu, theItem, DAName);
                            DaNum := OpenDeskAcc(DAName);
                            SetPort(oldPort);
                        END;
                END;
            END;
        END;
    FileID :
        BEGIN
            CASE theItem OF
                I_Quit :
                    doneFlag := TRUE;
            OTHERWISE
                BEGIN
                    END;
            END;
        END;
    EditID :
        BEGIN
            IF NOT SystemEdit(theItem - 1) THEN
                BEGIN
                    CASE theItem OF
                        I_Cut :
                            TECut(hte);
                        I_Copy :
                            TECopy(hte);
                        I_Paste :
                            TEPaste(hte);
                        I_Clear :
                            TDelete(hte);
                    OTHERWISE
                        BEGIN
                            END;
                        END;
                END;
            END;
        END;
    SearchID :
        BEGIN
            CASE theItem OF
                I_Find :

```

```

                Open_Find(I_Find);
            I_Replace :
                Open_Find(I_Replace);
            I_Show_Text :
                BEGIN
                    ShowWindow(MyWindow);
                    SelectWindow(MyWindow);
                END;
            OTHERWISE
                BEGIN
                    END;
                END;
            OTHERWISE
                BEGIN
                    END;
                END;
            END;
            HiliteMenu(0);
        END;

PROCEDURE FrameDialog;
VAR
    iBox : Rect;
    iType : integer;
    iHandle : Handle;
    oldPenState : PenState;
    oldPort : GrafPtr;
BEGIN
    GetPort(oldPort);
    SetPort(dlg);
    GetPenState(oldPenState);
    GetDialog(dlg, iNum, iType, iHandle, iBox);
    InsetRect(iBox, -4, -4);
    PenSize(3, 3);
    FrameRoundRect(iBox, 16, 16);
    SetPenState(oldPenState);
    SetPort(oldPort)
END;

PROCEDURE UpDate_Txt;
VAR
    oldPort : GrafPtr;
BEGIN
    GetPort(oldPort);
    SetPort(MyWindow);
    BeginUpDate(MyWindow);
    TEUpdate(hte^^.viewRect, hte);
    DrawGrowIcon(MyWindow);
    EndUpDate(MyWindow);
    SetPort(oldPort);
END;

PROCEDURE Act_Txt;
BEGIN
    SetPort(MyWindow);
    IF Odd(theEvent.modifiers) THEN
        BEGIN
            TEActivate(hte);
            DrawGrowIcon(MyWindow);
            DisableItem(EditMenu, I_Undo);
        END
    ELSE
        BEGIN
            TDeactivate(hte);
            DrawGrowIcon(MyWindow);
            EnableItem(EditMenu, I_Undo);
        END;
    END;
END;

PROCEDURE FixText;
BEGIN
    HLock(Handle(hte));
    WITH hte^^ DO
        BEGIN

```

```

viewRect := MyWindow^.portRect;
WITH viewRect DO
  BEGIN
    right := right - sBarWidth - 1;
    bottom := bottom - sBarWidth - 1;
    bottom := (bottom DIV lineHeight) * lineHeight;
  END;
destRect := viewRect;
InsetRect(destRect, TxtMargin, TxtMargin);
TECaltText(hte);
TESelView(hte);
END;
HUnlock(Handle(hte));
END;

PROCEDURE Grow_Txt;
BEGIN
  InvalRect(MyWindow^.portRect);
  EraseRect(MyWindow^.portRect);
  SizeWindow(MyWindow, hSize, vSize, true);
  InvalRect(MyWindow^.portRect);
  FixText;
END;

PROCEDURE Do_Quit;
BEGIN
  TEDispose(hte);
  DisposeWindow(MyWindow);
  ExitToShell;
END;
END.                                     (End of unit)

```

Listing: Find.Pas

```

UNIT Find_Dialog;
(File name:Find.Pas )
(Function: Handle a modeless dialog)
(History: 7/03/88 MEM - Made the Find dialog functional )
INTERFACE

```

```

  USES ROM85;

```

```

  PROCEDURE Init_Find;
  PROCEDURE Act_Find;
  PROCEDURE Open_Find (Action : integer);
  PROCEDURE Do_Find (theEvent : EventRecord; hte : THandle);
  PROCEDURE Close_Find;

```

IMPLEMENTATION

CONST

```

  FindID = 4;
  I_Find = 1;
  I_Replace = 2;
  I_Cancel = 3;
  I_Find_Label = 4;
  I_Find_Text = 5;
  I_Replace_Label = 6;
  I_Replace_Text = 7;
  Active = 0;
  InActive = 255;
  AlertID = 6;

```

VAR

```

  MyDialog : DialogPtr;
  ExitDialog : Boolean;
  tempRect : Rect;
  DType : Integer;
  Index : Integer;
  DItem : Handle;
  CItem, CTempItem : controlhandle;
  itemHit : Integer;
  temp : Integer;
  Bolden : integer;
  Opened : integer;
  theString : Str255;

```

```

PROCEDURE FrameItem (inum : integer);
VAR
  iBox : Rect;
  iType : integer;
  iHandle : Handle;
  oldPenState : PenState;
  oldPort : GrafPtr;
BEGIN
  GetPort(oldPort);
  SetPort(MyDialog);
  GetPenState(oldPenState);
  GetDItem(MyDialog, inum, iType, iHandle, iBox);
  InsetRect(iBox, -4, -4);
  PenSize(3, 3);
  FrameRoundRect(iBox, 16, 16);
  SetPenState(oldPenState);
  SetPort(oldPort)
END;

```

PROCEDURE DeFrameItem (inum : integer);

```

VAR
  iBox : Rect;
  iType : integer;
  iHandle : Handle;
  oldPenState : PenState;
  oldPort : GrafPtr;
BEGIN
  GetPort(oldPort);
  SetPort(MyDialog);
  GetPenState(oldPenState);
  GetDItem(MyDialog, inum, iType, iHandle, iBox);
  InsetRect(iBox, -4, -4);
  PenSize(3, 3);
  PenMode(PatBic);
  FrameRoundRect(iBox, 16, 16);
  SetPenState(oldPenState);
  SetPort(oldPort)
END;

```

PROCEDURE Init_Find;

```

BEGIN
  MyDialog := NIL;
END;

```

PROCEDURE Act_Find;

```

BEGIN
  IF (MyDialog <> NIL) THEN
    ShowWindow(MyDialog);
  END;

```

PROCEDURE Open_Find;

```

VAR
  ThisEditText : THandle;
  TheDialogPtr : DialogPtr;
BEGIN
  IF (MyDialog = NIL) THEN
    BEGIN
      MyDialog := GetNewDialog(FindID, NIL, Pointer(-1));
      ShowWindow(MyDialog);
      SelectWindow(MyDialog);
      SetPort(MyDialog);
      TheDialogPtr := DialogPeek(MyDialog);
      ThisEditText := TheDialogPtr^.textH;
      HLock(Handle(ThisEditText));
      ThisEditText^.txSize := 12;
      TextSize(12);
      ThisEditText^.txFont := systemFont;
      TextFont(systemFont);
      ThisEditText^.txFont := 0;
      ThisEditText^.fontAscent := 12;
      ThisEditText^.lineHeight := 12 + 3 + 1;
      HUnlock(Handle(ThisEditText));
      GetDItem(MyDialog, I_Replace_Text, DType, DItem,
tempRect);

```

```

        SetIText(DItem, '');
        SellText(MyDialog, I_Replace_Text, 0, 0);
        GetDItem(MyDialog, I_Find_Text, DType, DItem,
tempRect);
        SetIText(DItem, '');
        SellText(MyDialog, I_Find_Text, 0, 0);
        Opened := action;
        Bolden := I_Find;
    END
    ELSE
    BEGIN
        Opened := action;
        SelectWindow(MyDialog);
    END;
    IF (action = I_Find) THEN
    BEGIN
        DeFrameItem(I_Replace);
        HideDItem(MyDialog, I_Replace);
        HideDItem(MyDialog, I_Replace_Label);
        HideDItem(MyDialog, I_Replace_Text);
        GetDItem(MyDialog, I_Find, DType, DItem, tempRect);
        GetIText(DItem, theFindString);
        SetCTitle(ControlHandle(DItem), 'Find');
        SetWTitle(MyDialog, 'Find');
        Bolden := I_Find;
    END
    ELSE
    BEGIN
        ShowDItem(MyDialog, I_Replace);
        ShowDItem(MyDialog, I_Replace_Label);
        ShowDItem(MyDialog, I_Replace_Text);
        GetDItem(MyDialog, I_Replace, DType, DItem, tem-
pRect);
        HiLiteControl(ControlHandle(DItem), InActive);
        GetDItem(MyDialog, I_Find, DType, DItem, tempRect);
        SetCTitle(ControlHandle(DItem), 'Search');
        SetWTitle(MyDialog, 'Replace');
    END;
    DrawDialog(MyDialog);
    FrameItem(Bolden);
    ShowWindow(MyDialog);
    SelectWindow(MyDialog);
END;

PROCEDURE Close_Find;
BEGIN
    IF (MyDialog <> NIL) THEN
    BEGIN
        DisposDialog(MyDialog);
        Opened := 0;
        MyDialog := NIL
    END;
END;

FUNCTION Find (hte : TEHandle) : boolean;
VAR
    theFindString : Str255;
    StartAt, EndAt, targetlen, replacelen, FoundAt :
longint;
    destText : Handle;
    targetPtr, replacePtr : Ptr;
BEGIN
    Find := false;
    GetDItem(MyDialog, I_Find_Text, DType, DItem, tempRect);
    GetIText(DItem, theFindString);
    IF (theFindString = '') THEN
    BEGIN
        ParamText('You have not specified anything to Find',
'', '', '');
        temp := NoteAlert(AlertID, NIL);
    END
    ELSE
    BEGIN
        destText := Handle(hte^.hText);

```

```

        targetPtr := pointer(Ord(theFindString) + 1);
        replacePtr := NIL;
        IF (hte^.selStart < hte^.selEnd) THEN
            IF (opened = I_Find) THEN
                StartAt := hte^.selEnd
            ELSE
                StartAt := hte^.selStart
            ELSE
                StartAt := hte^.selStart;
                targetlen := length(theFindString);
                replacelen := 0;
                FoundAt := Munger(destText, StartAt, targetPtr,
targetlen, replacePtr, replacelen);
                IF (FoundAt >= 0) THEN
                BEGIN
                    TEsSetSelect(FoundAt, FoundAt + targetlen, hte);
                    TEsSelView(hte);
                    TEActivate(hte);
                    Find := true;
                END
            ELSE
                SysBeep(1);
            END;
        END;
    END;

    PROCEDURE Replace (hte : TEHandle);
    VAR
        theFindString, theReplString : Str255;
        StartAt, EndAt, targetlen, replacelen, FoundAt :
longint;
        destText : Handle;
        targetPtr, replacePtr : Ptr;
        OldPort : GrafPtr;
    BEGIN
        GetDItem(MyDialog, I_Find_Text, DType, DItem, tempRect);
        GetIText(DItem, theFindString);
        GetDItem(MyDialog, I_Replace_Text, DType, DItem, tem-
pRect);
        GetIText(DItem, theReplString);
        destText := Handle(hte^.hText);
        targetPtr := pointer(Ord(theFindString) + 1);
        targetlen := length(theFindString);
        replacePtr := pointer(Ord(theReplString) + 1);
        replacelen := length(theReplString);
        StartAt := hte^.selStart;
        FoundAt := Munger(destText, StartAt, targetPtr, targetlen,
replacePtr, replacelen);
        IF (FoundAt >= 0) THEN
        BEGIN
            TEsCallText(hte);
            TEsSetSelect(FoundAt, FoundAt, hte);
            TEsSelView(hte);
            GetPort(OldPort);
            SetPort(hte^.inPort);
            InvalRect(hte^.viewRect);
            SetPort(OldPort);
            TEActivate(hte);
            GetDItem(MyDialog, I_Replace, DType, DItem, tem-
pRect);
            HiLiteControl(ControlHandle(DItem), Inactive);
            DeFrameItem(I_Replace);
            FrameItem(I_Find);
            Bolden := I_Find;
        END
        ELSE
        BEGIN
            Sysbeep(1);
            GetDItem(MyDialog, I_Replace, DType, DItem, tem-
pRect);
            HiLiteControl(ControlHandle(DItem), Inactive);
            IF (bolden = I_Replace) THEN
                DeFrameItem(I_Replace);
            FrameItem(I_Find);
            Bolden := I_Find;

```

```

END;
END;

PROCEDURE Search (hte : THandle);
VAR
    theRep1String : Str255;
    success : boolean;
BEGIN
    GetDItem(MyDialog, I_Replace_Text, DType, Ditem, tempRect);
    GetIText(Ditem, theRep1String);
    success := Find(hte);
    IF success THEN
        BEGIN
            GetDItem(MyDialog, I_Replace, DType, Ditem, tempRect);
            HiLiteControl(ControlHandle(Ditem), Active);
            DeFrameItem(I_Find);
            FrameItem(I_Replace);
            Bolden := I_Replace;
        END;
    END;
END;

PROCEDURE Do_Find;
CONST
    ReturnKey = 13;
LABEL
    1;
VAR
    Index : integer;
    myPt : Point;
    ExitDialog : boolean;
    CmdDown : boolean;
    success : boolean;
    chCode : integer;
    MyCmdKey : char;
    whichDialog : DialogPtr;
BEGIN
    ExitDialog := false;
    IF (MyDialog <> NIL) THEN
        BEGIN
            IF (theEvent.what = KeyDown) THEN
                BEGIN
                    CmdDown := odd(theEvent.modifiers DIV CmdKey);
                    chCode := BitAnd(theEvent.message, CharCodeMask);
                    MyCmdKey := CHR(chCode);
                    IF CmdDown THEN
                        BEGIN
                            IF ((MyCmdKey = 'x') OR (MyCmdKey = 'X'))
THEN
                                DlgCut(MyDialog)
                            ELSE IF ((MyCmdKey = 'c') OR (MyCmdKey = 'C')) THEN
                                DlgCopy(MyDialog)
                            ELSE IF ((MyCmdKey = 'v') OR (MyCmdKey = 'V')) THEN
                                DlgPaste(MyDialog)
                            ELSE IF ((MyCmdKey = 'f') OR (MyCmdKey = 'F')) THEN
                                Open_Find(I_Find)
                            ELSE IF ((MyCmdKey = 'r') OR (MyCmdKey = 'R')) THEN
                                Open_Find(I_Replace);
                                GOTO 1
                            END
                        ELSE IF (chCode = ReturnKey) THEN
                            BEGIN
                                IF (bolden = I_Find) THEN
                                    IF (Opened = I_Find) THEN
                                        success := Find(hte)
                                    ELSE
                                        Search(hte)
                                    ELSE
                                        Replace(hte);
                                END
                            END
                        END
                    END
                END
            END
        END
    END

```

```

        GOTO 1;
    END;
    IF DialogSelect(theEvent, whichDialog, itemHit) THEN
        BEGIN
            GetDItem(whichDialog, itemHit, DType, Ditem, tempRect);
            Citem := Pointer(Ditem);
            IF (ItemHit = I_Cancel) THEN
                ExitDialog := true;
            IF (ItemHit = I_Find) THEN
                IF (Opened = I_Find) THEN
                    success := Find(hte)
                ELSE
                    Search(hte);
            IF (ItemHit = I_Replace) THEN
                Replace(hte);
            END
        ELSE IF (theEvent.what = UpDateEvt) THEN
            FrameItem(Bolden)
        ELSE IF (theEvent.what = ActivateEvt) AND (Opened = I_Find) AND Odd(theEvent.modifiers) THEN
            BEGIN
                GetDItem(MyDialog, I_Find, DType, Ditem, tempRect);
                GetIText(Ditem, theString);
                IF length(theString) > 0 THEN
                    SetIText(MyDialog, I_Find_Text, 0, length(theString));
                END;
            END;
        1 :
            IF ExitDialog THEN
                BEGIN
                    Close_Find;
                END;
            END;
        END;
    END.

```

Listing: Search.Pas

```

PROGRAM Searcher;
(Program name:Searcher.Pas )
(This is the main program and event loop. )
(History: 7/03/88 MEM Original Coding )
(
    USES
        ROM85, SearchGlobs, Inits, Find_Dialog, Utils;
    VAR
        theEvent : EventRecord;
        thecode : integer;
        theWindow : WindowPtr;
        thePeek : WindowPeek;
        OldPort : GrafPtr;
        menuResult : longint;
        newSize : longint;
        theMenu, theItem : integer;
        chCode : integer;
        ch : char;
        GlobalPt, localPt : Point;
        EventFlag : boolean;
        extend : boolean;
    BEGIN
        InitMac;
        InitRects;
        InitMenus;
        InitTxFind;
        Init_Find;
        REPEAT
            IF (hte <> NIL) THEN
                TEIdle(hte);
                SystemTask;
                EventFlag := GetNextEvent(everyEvent, theEvent);
                thecode := FindWindow(theEvent.where, theWindow);
            END
        UNTIL EventFlag = 0;
    END

```

```

IF IsDialogEvent(theEvent) THEN
  Do_Find(theEvent, hte)
ELSE IF EventFlag THEN
  BEGIN
    CASE theEvent.what OF
     MouseDown :
        BEGIN
          IF (thecode = inMenuBar) THEN
            BEGIN
              menuResult := MenuSelect(theEvent.Where);
              theMenu := HiWord(menuResult);
              theItem := LoWord(menuResult);
              Do_Menu(theMenu, theItem);
            END;
          IF (thecode = InDrag) THEN
            DragWindow(theWindow, theEvent.where,
DragRect);
          IF (thecode = inGrow) THEN
            BEGIN
              newSize := GrowWindow(theWindow,
theEvent.where, GrowRect);
              IF newSize <> 0 THEN
                Grow_Txt(LoWord(newSize),
HiWord(newSize))
              END;
            IF (thecode = inGoAway) THEN
              BEGIN
                IF TrackGoAway(theWindow, theEvent.where)
THEN
                  HideWindow(theWindow);
                END;
              IF ((thecode = inZoomIn) OR (thecode =
inZoomOut)) THEN
                BEGIN
                  IF TrackBox(theWindow, theEvent.where,
theCode) THEN
                    BEGIN
                      GetPort(OldPort);
                      SetPort(MyWindow);
                      EraseRect(MyWindow^.portRect);
                      ZoomWindow(MyWindow, thecode, TRUE);
                      WITH MyWindow^.portRect DO
                        Grow_Txt(right - left, bottom -
top);
                      SetPort(OldPort);
                    END;
                  END;
                IF (thecode = inContent) THEN
                  BEGIN
                    IF (theWindow <> FrontWindow) THEN
                      SelectWindow(theWindow)
                    ELSE
                      BEGIN
                        globalPt := theEvent.where;
                        localPt := globalPt;
                        GlobalToLocal(localPt);

```

```

THEN
  IF PtInRect(localPt, hte^.viewRect)
  BEGIN
    extend :=
  (BitAnd(theEvent.modifiers, ShiftKey) <> 0);
    TEClick(localPt, extend, hte);
  END;
  END;
  IF (thecode = inSysWindow) THEN
    SystemClick(theEvent, theWindow);
  END;
  KeyDown, AutoKey :
  BEGIN
    WITH theEvent DO
      BEGIN
        chCode := BitAnd(message, CharCodeMask);
        ch := CHR(chCode);
        IF (Odd(modifiers DIV CmdKey)) THEN
          BEGIN
            menuResult := MenuKey(ch);
            theMenu := HiWord(menuResult);
            theItem := LoWord(menuResult);
            IF (theMenu <> 0) THEN
              Do_Menu(theMenu, theItem)
            ELSE IF ((ch = 'x') OR (ch = 'X')) AND
(chte <> NIL) THEN
              TECut(hte)
            ELSE IF ((ch = 'c') OR (ch = 'C')) AND
(chte <> NIL) THEN
              TECopy(hte)
            ELSE IF ((ch = 'v') OR (ch = 'V')) AND
(chte <> NIL) THEN
              TEPaste(hte);
            END
          ELSE IF (hte <> NIL) THEN
            TEKey(ch, hte);
          END;
        END;
      UpDateEvt :
      BEGIN
        Update_Txt;
      END;
      ActivateEvt :
      BEGIN
        Act_Txt(theEvent);
      END;
    OTHERWISE
      BEGIN
        END;
      END;
    UNTIL doneFlag;
    Do_Quit;
  END.

```



Pascal Procedures

Have Your Window Do The Splits



Kirk Chase
Anaheim, CA
Volume 5, Number 6

Split Personality

As I was looking in chapter two of *Inside Macintosh*, Vol. I, I saw a sweet little interface under the heading "Splitting a Window." I thought about that interface and decided it would be nice to create a split bar in a window. So, I set out to write a control definition for the split bar and a short little demo to see what additional support was needed by the application.

The Split Bar Interface

A split bar is used to divide a window into two or more views of a document. The split bar creates and readjusts scroll bars on top and bottom of the split bar as it is moved. This allows the user to scroll each view, called a "pane," to different areas (See Figure 1). Many word processing and spreadsheet applications use split bars.

For example, at program start up, there is one scroll bar and a black rectangle (the split bar) directly above it and beneath the drag bar of the window. The user then clicks on the split bar and drags it across the top of the scroll bar. As he releases the split bar, the scroll bar's top is brought down on the screen just under the split bar again and another scroll bar above the split bar fills the space left by moving the split bar. As the user continues to move the split bar up or down, the top and bottom scroll bars are adjusted in height accordingly.

The Split Bar CDEF

The split bar functions like a scroll bar with only a thumb. In some ways, it is easier than a normal scroll bar to create. For example, there is no concern for page up/down or line up/down—only a thumb routine (which is not very easy until you know how). I include two variations of the CDEF for both horizontal (1) and vertical (2) split bars.

The CDEF for a split bar is somewhere between the complexity of a normal button and a scroll bar. If you read up on defining your own controls (*Inside Macintosh*, Vol. I, Chapter 10), there are nine routines that your control function may or may not need to handle. Most of the routines I, and others, have covered before in previous issues of *MacTutor* (Vol. 5, No. 1 and 4). I will only examine those parts that apply to the split bar.

calcCRgns Message

This message asks the CDEF to calculate the region of the control or indicators and store it in the region handle supplied in the *param* parameter. Upon entry, you test *param* to see if the high bit is set. If it is, you give the region for the indicator, and if not, you give the region of the control.

The region of the control is its enclosing rectangle. This rectangle extends the length of the window. This area is the same,

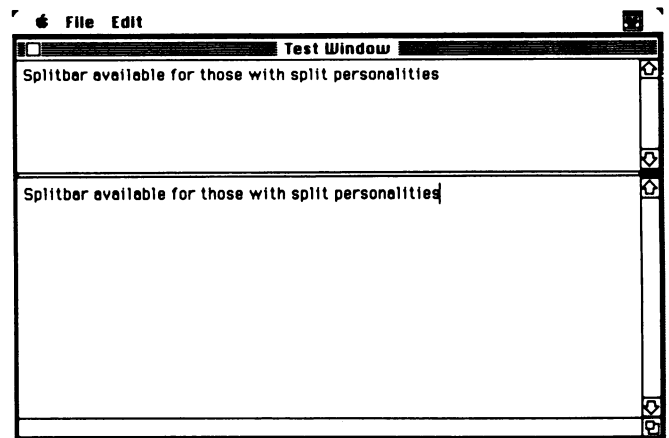


Figure 1. Demo Application

except for a few areas, as the area the two scroll bars occupy. (See Figure 2).

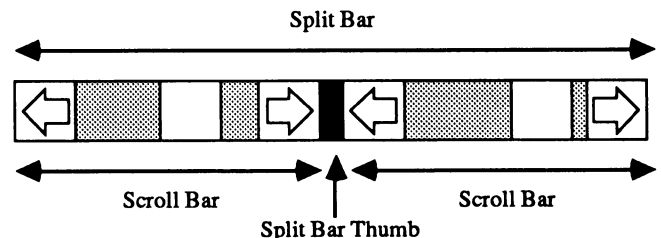


Figure 2 Scroll and Split Bars

Stacking controls is not a good idea (how do you know which control you are in?). But, since the Split Bar CDEF only reports a mouse down in the indicator, which is not covered, we have no problem knowing which control was pressed.

Now if the message wants the region of the thumb indicator, it is because it is about to be dragged around the screen with *DragGrayRgn*. Rather than just the region of the thumb, I decided to give additional, visual interface to the user by dragging a rectangular gray region across the window along with the thumb. In the application, I draw a couple lines across the screen to separate the panes, and this gives the feel that these two lines are part of the control. Calculating this other region is easy since all points are known.

thumbCntl Message

This message is another routine used as a precursor to dragging an image of the thumb. The *param* parameter is a pointer to the following record:

```

thumbinfo = record;
  limitRect : Rect;
  trackRect : Rect;
  axis : integer;
end;

```

This record has the same meaning as that of the DragControl procedure. `limitRect` confines the movement of the indicator itself; `trackRect` defines the rectangle where the indicator will track with the mouse (these two are just the control's rectangle); and `axis` determines tracking constraints—1 for horizontal, 2 for vertical, and 0 for none. Notice that the axis constraints are the same as the variation of the control (I did not implement a 0 variation).

posCntl Message

After the thumb has been dragged all over, the CDEF is called again with a `posCntl` message to reposition the thumb. You must erase the old position of the indicator which I do with a flag sent to my draw routine and invalidating the place where the indicator was.

The next step is to figure out the new value of the control. Our friend, *param*, is typecasted to a point and holds the relative offset both horizontal and vertical. The new value of the control is the old value plus the offset, minus any amount that goes out of bounds. Finally, the control is drawn at its new value.

Other Messages

The `initCntl` message just sets the `cntlAction` field of the control to nil. The `drawCntl` message holds nothing mysterious that hasn't been covered before. There is nothing done for the `dispCntl`, `dragCntl`, and `autoTrack` messages. The latter two messages are for further customizing your control.

The Split Bar Demo

I decided to write a simple Text Edit demo using a vertical split bar. The user could pull the split bar up and down, resize the window, scroll each pane independently, do editing in either pane, and see updates if applicable in both panes. This was certainly a mouthful.

I tried to think about how to handle text in the two panes. I could keep two Text Edit records identical except for view and destination rectangles, or I could use one record and just switch the view and destination rectangles when needed. I decided to implement the latter version. Although separate TE records would have made it simpler to develop and not taken too much memory, I thought that this demo could have easily been a spreadsheet or a drawing application where the document might be too much to duplicate (what if there were a possibility of four or even nine panes?).

The next step was to readjust the view rectangles and scroll bars when the split bar was moved. Since the value of the split bar control was in pixels, this presented no problems. I just adjusted the bottom of the top TE view rectangle by the split bar value and the lower TE view rectangle by split bar value and the indicator width. I also did the same with the scroll bars. In fact, I used the same routines to fix the view rectangles and scroll bar

heights when the window was grown as when the split bar was moved.

There was the problem of when the height of the scroll bar was less than what was needed to draw the scroll bar properly. So if the height was smaller, I made the control invisible. I also restricted the growing of the window as to leave a minimum space from the split bar. Therefore, if the height did not allow a scroll bar to be drawn correctly, it was not drawn, and the user could not scroll that pane.

Besides responding to the grow and split bar, each pane must respond to editing such as cut, copy, paste, and keyboard input. The standard procedure is used in the active pane and then the other pane is invalidated to take care of updating in the update procedure.

The pane where the action such as scrolling and editing happens is quite easy to figure out. The scroll bars have a pane associated with it. They switch the TE record to the proper pane, scroll the TE record, and then reset the TE record back to the active pane. Mouse downs in the content region see which pane they are in and then switch the active pane. I also do automatic scrolling to the selection for the active pane only (the non active pane only moves when scrolled by its scroll bar or by major editing).

The updating of the window is easy to accomplish. Simply switch panes and do a TEUpdate call to both panes.

That is about it, the demo is pretty much your standard TE demo you have seen before. You simply switch the view and destination rectangles, duplicate or update the other, and then switch back to the active pane. You will also notice several TEActivates and TEDeactivates throughout the source code. I found that the blinking insertion point would stay around if you switched between panes or typed while it was drawn. The solution was to turn it off, switch panes or edit, and then turn it back on.

Improvements

I limited the demo to a single, vertical split bar. To improve the demo, you could include a horizontal split bar making a possibility of four panes. You might even want to add multiple split bars to the horizontal and/or vertical dimensions. This could give you to nine panes if you have two horizontal and two vertical split bars. There would be the problem of keeping a split bar invisible if it was covered by another split bar.

There is one minor bug I could not track down. On starting the demo after the Mac has been restarted, the indicator's outline is not drawn when it is dragged. However, if you click on the drag bar of the window, creating a window outline, then the indicator outline appears from then on, until the Mac is restarted. I tried fixing this but could not do so. If any of the readers have any insight, please write me at MacTutor, and I'll relay the information in the letters column.

The CDEF allows horizontal and vertical variations (1 and 2). These numbers follow the axis constraints for dragging the thumb. How about a combined horizontal and vertical split bar which would have a variation code of 0 (no constraint)? This could be used for maybe a new and improved combined scrolling.

The CDEF could be used for other things such as T bars for drawing programs.

All this is left as an “exercise to the reader.” Good luck! Now your windows will have the same type of personality as the developer—split.

Listing: splitbar.pas

```
unit MyControl;
(Splitbar Code Definition Function - ID=17)
(This creates two types of splitbar controls - horizontal,
 ( variation code 1; and vertical, variation code 2)
 (A Splitbar is essentially just an indicator (thumb) which
 ( can be moved by the mouse to set )
 (up window panes. The control only moves the thumb. It is)
 ( up to the application to create/resize normal)
 (scrollbars, adjust the content region, and so forth. It )
 (will only return the Indicator part code of)
 (inThumb (129). There are no page/line up/down parts)
 (valid min is 0 and max is screen width - indicatorwidth;)
 ( control value is then in pixels)
 (To get a horizontal splitbar ask for CDEF 273 ( 16*ID + )
 ( variation), and 274 for vertical)
 (History)
 (3/15/89 Created by Kirk Chase)

interface
 ( main entry into CDEF )
 function main (varCode: integer; theControl: ControlHandle;
 message: integer; param: longint): longint;

implementation
const
 vSplitBar = 2; (Variation code for a vertical splitbar)
 hSplitBar = 1; (Variation code for a horizontal splitbar)
 IndicatorWidth = 6; (width of thumb)
 PaneWidth = 4;
 draw = 1;
 erase = 0;
 invisible = 0;
 inactive = 255;

function main;

  procedure doRect (varcode, value: integer; var theRect:
 rect);
 (calculate indicator rectangle according to varcode)
 begin
  case varcode of
   vSplitBar:
    begin
     theRect.top := value + theRect.top;
     theRect.bottom := theRect.top + IndicatorWidth;
     InsetRect(theRect, 1, 0);
    end;
   hSplitBar:
    begin
     theRect.left := value + theRect.left;
     theRect.right := theRect.left + IndicatorWidth;
     InsetRect(theRect, 0, 1);
    end;
  end;
 end;

  procedure doInit (myControl: ControlHandle);
 (inits control by setting the action proc to nil)
 begin
  myControl^.ctrlAction := nil; (set action proc )
 end; (of doInit)

  procedure doDraw (varCode: integer; myControl: Control-
 Handle; flag: integer);
```

```
(this will draw or erase thumb control according to flag)
var
 aRect, iRect: Rect;
 oldClip, controlRegion: RgnHandle;
 oldPen: PenState;
begin
 (only draw if visible)
 if (myControl^.ctrlVis <> invisible) then
  begin
   ( Get control's region & set clip region to region.)
   oldClip := NewRgn;
   GetClip(oldClip);

   ( Set the clip region to the control's rectangle )
   aRect := myControl^.ctrlRect;
   iRect := aRect;
   controlRegion := NewRgn;
   RectRgn(controlRegion, aRect);
   MoveHi(Handle(myControl));
   HLock(Handle(myControl));
   SetClip(controlRegion);
   HUnlock(Handle(myControl));

   (set pen to normal state)
   GetPenState(oldPen);
   PenNormal;

   FrameRect(aRect); (draw control bounds)

   doRect(varcode, myControl^.ctrlValue, iRect);

   (either draw or erase indicator)
   if flag = draw then
    PaintRect(iRect)
   else
    EraseRect(iRect);

   if (myControl^.ctrlHilite = inactive) then
    EraseRect(iRect); (inactive controls)

   SetClip(oldClip); (Clean up)
   DisposeRgn(oldClip);
   DisposeRgn(controlRegion);
   SetPenState(oldPen);
  end;
end; (of doDraw)

function doTest (varcode: integer; myControl: Control-
 Handle; theParam: longint): longint;
 (returns inThumb or 0 if mousedown in thumb or not)
var
 CRect, IRect: Rect;
 thePoint: point;
begin
 CRect := myControl^.ctrlRect; (initialize values)
 IRect := CRect;
 thePoint := point(theParam);
 doTest := 0;

 (test point if active and visible)
 if (myControl^.ctrlHilite <> inactive) and
 (myControl^.ctrlVis <> invisible) then
  begin
   (in control?)
   if PtInRect(thePoint, CRect) then
    begin
     (in thumb?)
     doRect(varcode, myControl^.ctrlValue,
 IRect);

     if PtInRect(thePoint, IRect) then
      doTest := inThumb;
    end;
  end;
end; (of doTest)
```

```

procedure doCalc (varcode: integer; myControl: Control-
Handle; theParam: longint);
  (calculate all or indicator's region)
var
  aRect: Rect;
  thumbRgn: RgnHandle;
begin
  ( CalcButtnRgn must first find out if high bit is set. )
  ( High bit set indicates that region being calculated is )
  ( for an indicator )
  if not BitTst(Ptr@theParam, 0) then
    begin (whole region)
      theParam := longint(BitAnd(theParam, $00FFFFFF));
      aRect := myControl^.ctrlRect;
      RectRgn(RgnHandle(theParam), aRect);
    end
  else
    begin
      aRect := myControl^.ctrlRect;
      doRect(varcode, myControl^.ctrlValue, aRect);
      thumbRgn := NewRgn;
      RectRgn(thumbRgn, aRect);
      if varcode = vSplitBar then (get rgn )
        SetRect(aRect, 0, aRect.top + 1, aRect.right,
aRect.bottom - 1) (vertical splitbar)
      else
        SetRect(aRect, aRect.left + 1, 0, aRect.right -
1, aRect.top); (horizontal splitbar)
        RectRgn(RgnHandle(theParam), aRect);
        UnionRgn(RgnHandle(theParam), thumbRgn,
RgnHandle(theParam));
        DisposeRgn(thumbRgn);
      end;
    end; (of doCalc)

procedure doThumb (myControl: ControlHandle; varcode:
integer; theParam: longint);
  (this sets up dragging parameters for thumb)
  type
    thumbPtr = ^thumbInfo;
    thumbInfo = record
      limitRect: Rect;
      trackRect: Rect;
      axis: integer;
    end;
  begin
    with thumbPtr(theParam) do
      begin
        limitRect := myControl^.ctrlRect;
        trackRect := myControl^.ctrlRect;
        axis := varcode;
      end;
    end; (of doThumb)

procedure doPosition (myControl: ControlHandle; varcode:
integer; DeltaPoint: longint);
  (this routine is called to reposition the control )
  (first erase old position of control and draw in new place)
  var
    thePoint: point;
    value, delta, position: integer;
    aRect: rect;
  begin
    aRect := myControl^.ctrlRect; (get thumb region)
    doRect(varcode, myControl^.ctrlValue, aRect);
    InvalRect(aRect);
    doDraw(varCode, myControl, erase); (erase)

    thePoint := point(DeltaPoint);
    value := myControl^.ctrlValue;

    if varcode = vSplitBar then (calculate delta offset)
      begin

```

```

        position := value + thePoint.v;
        delta := thePoint.v;
      end
    else
      begin
        position := value + thePoint.h;
        delta := thePoint.h;
      end;

    (recalculate delta offset if out of bounds)
    if position < myControl^.ctrlMin then
      delta := -(value - myControl^.ctrlMin);
    if position > myControl^.ctrlMax then
      delta := myControl^.ctrlMax - value;

    myControl^.ctrlValue := myControl^.ctrlValue +
delta; (reset control value)

    doDraw(varCode, myControl, draw); (redraw)
  end; (of doPosition)

begin (main entry point)
  main := 0; (initialize values)
  case message of (switch to proper routine)
    initCntl:
      doInit(theControl);

    drawCntl:
      doDraw(varCode, theControl, draw);

    testCntl:
      main := doTest(varcode, theControl, param);

    ( Calc the region for the button. )
    calcCRgns:
      doCalc(varcode, theControl, param);

    thumbCntl:
      doThumb(theControl, varcode, param);

    posCntl:
      doPosition(theControl, varcode, param);

    ( Nothing to do for these messages... )
    dragCntl, autoTrack, dispCntl:
      ;
    otherwise
      end;
  end;
end. (of MyControl Unit)

```

Listing: MyGlobals.pas

```

unit MyGlobals;
interface
  const
    Splitbar = 10; (Scroll bar ID)
    Scrollbar2 = 9; (Scroll bar ID)
    Scrollbar1 = 8; (Scroll bar ID)
    SBarWidth = 16; (Scrollbar Width)
    AppleMenuID = 1; (Menu list)
    About_Splitbar = 1;
    FileMenuID = 2; (Menu list)
    C_New = 1;
    C_Close = 3;
    C_Quit = 5;
    EditMenuID = 3; (Menu list)
    C_Undo = 1;
    C_Cut = 3;
    C_Copy = 4;
    C_Paste = 5;
    SBarMinLen = 55; (Min. Scroll bar hieght to be usable)
    CR = Chr(13);

```

```

VSpiltbar = 274; (variation code for split bar)
IndicatorWidth = 6; (split bar width)
Margin = 4; (margin inset for TE)
AboutDialogID = 1; (about dialog ID)
WindowID = 2; (MyWindow ID)
About_OK = 1; (OK button in About Dialog)
invisible = 0;
visible = 255;
inactive = 255;
active = 0;

mBarHeightGlobal = $BAA; (location of menu bar height)
WNETrapNum = $60;
UnImplTrapNum = $9F;
MultiEvt = 15;
bit0 = 31;
GrayRgnLowMemGlobal = $9EE;

var
  Scroll: array[1..2] of ControlHandle; (scrollbars)
  Split: ControlHandle; (splitbar control)

  MyWindow: WindowPtr;

  viewRect: array[1..2] of rect; (TE viewRects switching)
  destRect: array[1..2] of rect; (TE destRects switching)
  activePane: integer; (active pane where insertion pt is)

  EditText: TEHandle; (the TE record for the editing)

  paneRect: rect; (the two lines that split the pane)

  AppleMenuHandle, FileMenuHandle, EditMenuHandle:
  MenuHandle; (various menu handles)

  theWorld: SysEnvRec;
  theErr: OSErr;
  WNE: boolean;
  mBarHeight: integer;
implementation
end.

```

Listing: MyScroll.pas

```

unit MyScroll;
interface
uses
  MyGlobals;

procedure AdjustScrollBars; (reset max of scroll bars)
procedure AdjustText (pane: integer); (scroll to value)
procedure ScrollCharacter; (scroll selection into view)
procedure ScrollToSelection; (figure which way to scroll)
procedure SetEditText (Index: integer); (Switch TE pane)
procedure FixTextRects; (resize EditText's rectangles)
procedure FixScrollbarRects; (resize & post scroll/split )
procedure UpdateOtherPane; (inval nonactivepane's update )

```

implementation

```

( This is taken from Symantec's Think's Pascal)
function LinesInText: integer;
var
  lines: integer;
  txt: CharsHandle;
begin
  with EditText^^ do
    begin
      lines := nLines;
      txt := CharsHandle(hText);
      if teLength > 0 then
        if txt^^[teLength - 1] = CR then (Return?)
          lines := lines + 1;
      LinesInText := lines
    end

```

```

    end
  end; (of LinesInText)

procedure AdjustScrollBars; (reset max of scroll bars)
var
  ctlMax, index, height: integer;
begin
  for index := 1 to 2 do (for each pane)
    begin
      SetEditText(index); (set pane)
      with EditText^^ do
        begin
          height := (viewRect.bottom - viewRect.top) div
lineHeight; (lines of text in viewRect)
          ctlMax := LinesInText - height; (Control max)
          if ctlMax < 0 then
            ctlMax := 0;
          SetCtlMax(Scroll[index], ctlMax)
        end
      end;
      SetEditText(activePane); (reset back to active pane)
    end; (of AdjustScrollBars)

procedure AdjustText; (scroll to scrollbar value)
var
  oldScroll, newScroll, delta: integer;
begin
  SetEditText(pane);
  with EditText^^ do
    begin
      oldScroll := viewRect.top - destRect.top; (get old )
      newScroll := GetCtlValue(Scroll[pane]) * lineHeight;
        (get new scroll value)
      delta := oldScroll - newScroll;
      if delta <> 0 then
        begin
          TEScroll(0, delta, EditText); (scroll to new )
        end;
      end;
      destRect[pane] := EditText^^.destRect; (reset destRect )
      SetEditText(activePane); (reset to active pane)
    end; (of AdjustText)

procedure ScrollCharacter; (scroll selection into view)
var
  theLine, height: integer;
begin
  HLock(Handle(EditText));
  theLine := 0;
  with EditText^^ do
    begin
      height := (viewRect.bottom - viewRect.top) div
lineHeight;
      while selStart >= lineStarts[theLine] do
        theLine := theLine + 1;
      SetCtlValue(Scroll[activePane], theLine - nLines div
2);
      AdjustText(activePane);
    end;
  HUnlock(Handle(EditText));
  end; (of scroll character)

procedure ScrollToSelection; (way to scroll to select)
var
  topline, bottomline, height, max: integer;
begin
  SetEditText(activePane);
  HLock(Handle(EditText));
  AdjustScrollBars;
  AdjustText(activePane);
  with EditText^^, viewRect do
    begin
      topline := GetCtlValue(Scroll[activePane]);
      height := (bottom - top) div lineHeight;

```

```

        bottomLine := topline + height;
        max := GetCtlMax(Scroll[activePane]);
        if max = 0 then (all text within pane)
            AdjustText(activePane)
        else
            ScrollCharacter; (need to scroll)
        end;
        HUnlock(Handle(EditText));
    end; (of ScrollToSelection)

procedure SetEditText;
(Set EditText to indexed pane)
begin
    EditText^.viewRect := viewRect[Index];
    EditText^.destRect := destRect[Index];
end; (of SetEditText)

procedure FixTextRects; (adjust EditText's two panes)
begin
    HLock(Handle(EditText));
    viewRect[1] := MyWindow^.portRect; (set lower pane)
    viewRect[1].top := Split^.ctrlValue + IndicatorWidth;
    viewRect[1].right := viewRect[1].right - SBarWidth;
    viewRect[1].bottom := viewRect[1].bottom - SBarWidth;
    InsetRect(viewRect[1], Margin, Margin); (Give TE margins)

    (Make viewRect a Multiple of lineHeight)
    viewRect[1].bottom := viewRect[1].bottom -
    ((viewRect[1].bottom - viewRect[1].top) mod
    EditText^.lineHeight);
    destRect[1] := viewRect[1]; (Set destRect)

    viewRect[2] := MyWindow^.portRect; (set upper pane)
    viewRect[2].bottom := Scroll[2]^^.ctrlRect.bottom;
    viewRect[2].right := viewRect[2].right - SBarWidth;
    InsetRect(viewRect[2], Margin, Margin); (Give TE margins)

    (Make viewRect a multiple of lineHeight)
    viewRect[2].bottom := viewRect[2].bottom -
    ((viewRect[2].bottom - viewRect[2].top) mod
    EditText^.lineHeight);
    destRect[2] := viewRect[2];

    SetEditText(activePane); (Reset back to active pane)

    if EditText <> nil then (recalculate line starts)
        begin
            TEdtText(EditText);
            AdjustText(1);
            AdjustText(2);
            ScrollToSelection;
        end;
        HUnlock(Handle(EditText));
    end; (of FixTextRects)

procedure FixScrollbarRects; (Fix scrollbars to splitbar)
    var
        width, hieght: integer;
        temp: rect;
    begin
        with MyWindow^.portRect do (get window dimensions)
            begin
                hieght := bottom - top - SBarWidth + 1;
                width := right - left - SBarWidth + 1;
            end;

        SetRect(temp, width, 0, width + SBarWidth, hieght);
        Split^.ctrlRect := temp;
        Scroll[1]^^.ctrlRect := Split^.ctrlRect;
        Scroll[2]^^.ctrlRect := Split^.ctrlRect;
        Scroll[1]^^.ctrlRect.top := Split^.ctrlValue +
IndicatorWidth;
        Scroll[2]^^.ctrlRect.bottom := Split^.ctrlValue;

```

```

        (check to see if there is enough room to draw scroll bars)
        if (Scroll[2]^^.ctrlRect.bottom -
        Scroll[2]^^.ctrlRect.top) < SBarMinLen then
            begin (not enough room, make invisible)
                EraseRect(Scroll[2]^^.ctrlRect);
                Scroll[2]^^.ctrlVis := invisible;
            end
        else (enough room, make visible)
            Scroll[2]^^.ctrlVis := visible;

        if (Scroll[1]^^.ctrlRect.bottom -
        Scroll[1]^^.ctrlRect.top) < SBarMinLen then
            begin (not enough room, make invisible)
                EraseRect(Scroll[1]^^.ctrlRect);
                Scroll[1]^^.ctrlVis := invisible;
            end
        else (enough room, make visible)
            Scroll[1]^^.ctrlVis := visible;

        if EditText <> nil then (fix TE rectangles)
            begin
                FixTextRects;
                EraseRect(MyWindow^.portRect);
                InvalRect(MyWindow^.portRect);
            end;
        end; (of FixScrollbarRects)

procedure UpdateOtherPane; (invalidate non active pane)
    var
        tempRect: array[1..2] of rect;
        delta: array[1..2] of integer;
        i, index: integer;
        pane: rect;
        intersect: boolean;
    begin
        case activePane of (get non active pane)
            1:
                index := 2;
            2:
                index := 1;
        end;

        for i := 1 to 2 do (normalize viewRects)
            begin
                delta[i] := destRect[i].top;
                tempRect[i] := viewRect[i];
                OffsetRect(tempRect[i], 0, -delta[i]);
            end;

        intersect := SectRect(tempRect[1], tempRect[2], pane);
        (get intersection)
        OffsetRect(pane, 0, delta[index]); (place intersection pt)

        if intersect then (invalidate intersecting rectangle)
            InvalRect(pane);
        end; (of UpdateOtherPane)

end.

```

Listing: Test_Window.pas

```

unit Test_Window;

```

```

    (File name: Test_Window.Pas)
    (Function: Handle a Window)

```

interface

```

uses
    MyGlobals, MyScroll;

```

```

procedure Close_Test_Window (whichWindow: WindowPtr);
    (Close our window)

```

```

procedure Open_Test_Window; (Open our window)

```

```

procedure Update_Test_Window; (Update our window)

```

```

procedure Do_Test_Window (myEvent: EventRecord);
    (Handle action to our window, like controls)

implementation
    (=====)
    procedure Close_Test_Window; (Close our window)
    begin
        if (MyWindow <> nil) and (MyWindow = whichWindow) then
            begin
                DisposeWindow(MyWindow); (Clear window and controls)
                MyWindow := nil;
                TEDispose(EditText); (dispose of TE record)
                EditText := nil;
            end;
            (End for if (MyWindow<>nil))
        end; (of Close_Test_Window)

    procedure UpDate_Test_Window; (Update our window)
    begin
        PenNormal;
        EraseRect(paneRect); (redraw dividing lines two panes)
        SetRect(paneRect, 0, Split^.ctrlValue + 1,
        Split^.ctrlRect.left, Split^.ctrlValue + 5);
        if ((Split^.ctrlValue > 0) and (Split^.ctrlValue <
        Split^.ctrlMax)) then (is there enough room?)
            begin
                FrameRect(paneRect);
            end;

        if (EditText <> nil) then (update EditText to both panes)
            begin
                TEDeactivate(EditText);
                SetEditText(1);
                TEUpdate(MyWindow^.portRect, EditText); (Lower pane)
                SetEditText(2);
                TEUpdate(MyWindow^.portRect, EditText); (Upper pane)
                SetEditText(activePane);
                TEActivate(EditText);
            end;

        DrawControls(MyWindow); (Draw all the controls)
        DrawGrowIcon(MyWindow); (Draw the Grow box)
    end; (of UpDate_Test_Window)

    procedure Open_Test_Window; (Open our window and draw)
    var
        sTemp: str255; (For initializing the text window)
    begin
        if (MyWindow = nil) then
            begin
                MyWindow := GetNewWindow(WindowID, nil, Pointer(-1));
                SelectWindow(MyWindow); (Bring our window to front)
                SetPort(MyWindow);

                ( Make a scroll bars)
                Scroll[2] := GetNewControl(Scrollbar2, MyWindow);
                Scroll[1] := GetNewControl(Scrollbar1, MyWindow);
                ( Make a splitbar)
                Split := GetNewControl(Splitbar, MyWindow);

                (Set max/min and initial values of scrollbars)
                SetCtlMax(Scroll[1], 0);
                SetCtlMin(Scroll[1], 0);
                SetCtlValue(Scroll[1], 0);
                SetCtlMax(Scroll[2], 0);
                SetCtlMin(Scroll[2], 0);
                SetCtlValue(Scroll[2], 0);
                FixScrollbarRects; (redraw scroll/split bars done)

                FixTextRects; (get TE record's rects for switching)

                activePane := 1; (Set active pane)

                EditText := TENew(viewRect[1], destRect[1]);
                HLock(Handle(EditText)); ( setup attributes of TE )
            end
        end
    end

```

```

        with EditText do
            begin
                txFont := app1Font;
                fontAscent := 12;
                lineHeight := 12 + 3 + 1;
            end;
            HUnLock(Handle(EditText));
            sTemp := 'Splitbar available for those with split
            personalities';
            TEText(Pointer(Ord4(0sTemp) + 1), length(sTemp),
            EditText);
            FixTextRects;
            TEActivate(EditText);
            ShowWindow(MyWindow);
            UpDate_Test_Window; (Do update to draw rest)
        end
    else
        SelectWindow(MyWindow); (Already open, so show it)
    end; (of Open_Test_Window)

    procedure HandleScrollBars (theControl: ControlHandle;
    thePart: integer); (scroll Text while in scrollbar)
    var
        oldValue, delta, pane: integer;
    begin
        case GetRefCon(theControl) of (find which pane)
            ScrollBar1:
                pane := 1;
            ScrollBar2:
                pane := 2;
        end;

        case thePart of (get delta)
            inUpButton:
                delta := -1;
            inDownButton:
                delta := 1;
            inPageUp:
                delta := -(viewRect[pane].bottom -
                viewRect[pane].top) div EditText^.lineHeight;
            inPageDown:
                delta := (viewRect[pane].bottom - viewRect[pane].top)
                div EditText^.lineHeight;
        end;

        if thePart <> 0 then (set Control's value & adjust text)
            begin
                oldValue := GetCtlValue(theControl);
                SetCtlValue(theControl, oldValue + delta);
                AdjustText(pane);
            end;
        end; (of HandleScrollBars)

    procedure Do_Test_Window; (Handle action to our window)
    var
        RefCon: integer;
        code: integer; (Location of event in window or controls)
        theValue: integer; (Current value of a control)
        whichWindow: WindowPtr;
        myPt: Point; (Point where event happened)
        theControl: ControlHandle; (Handle for a control)
        extend: boolean; (TE extending with Shift key modifier)

    procedure Do_A_ScrollBar (code: integer); (do ScrollBar)
    begin
        RefCon := GetRefCon(theControl); (get control refcon)

        case RefCon of (Select correct scrollbar)
            Splitbar: (Splitbar)
                begin (start for this scrollbar)
                    TEDeactivate(EditText);
                    code := TrackControl(theControl, myPt, nil);
                    FixScrollbarRects;
                    TEActivate(EditText);
                end
        end
    end

```

```

end;

Scrollbar1:
begin (start for this scroll bar)
  if code <> inThumb then
    code := TrackControl(theControl, myPt, @HandleScrollBars)
  else
    begin
      code := TrackControl(theControl, myPt, nil);
      AdjustText(1);
    end;
  end;

Scrollbar2:
begin (start for this scroll bar)
  if code <> inThumb then
    code := TrackControl(theControl, myPt, @HandleScrollBars)
  else
    begin
      code := TrackControl(theControl, myPt, nil);
      AdjustText(2);
    end;
  end;
end; (end of case)
end; (Handle a ScrollBar being pressed)

begin (Start of Window handler)
  if (MyWindow <> nil) then
    begin
      code := FindWindow(myEvent.where, whichWindow);

      if (myEvent.what =MouseDown) and (MyWindow =
whichWindow) then (convert coords)
        begin
          myPt := myEvent.where;
          GlobalToLocal(myPt);
        end;

      if (MyWindow = whichWindow) and (code = inContent)
then (for our window)
        begin
          TEDeactivate(EditText); (remove highlighting)
          if PtInRect(MyPt, viewRect[1]) then
            (determine which pane mouse down in)
            begin
              SetEditText(1);
              activePane := 1;
            end
          else if PtInRect(MyPt, viewRect[2]) then
            begin
              SetEditText(2);
              activePane := 2;
            end;
          TEActivate(EditText); (put highlighting back)

          if PtInRect(MyPt, EditText^.viewRect) then
            begin
              extend := (BitAnd(myEvent.modifiers,
ShiftKey) <> 0);
              TEClick(myPt, extend, EditText);
            end;

          code := FindControl(myPt, whichWindow, theControl); (Get type of control)
          if (code <> 0) then (Check type of control)
            code := TrackControl(theControl, myPt, nil);

          if (code = inUpButton) or (code = inDownButton)
or (code = inThumb) or (code = inPageDown) or (code =
inPageUp) then
            Do_A_ScrollBar(code); (Do scrollbars)

```

```

end;
end;
End of procedure)

end.
(End of unit)

```

Listing: SplitbarTest.pas

```

program SplitbarTest;
  (Program name: SplitbarTest.Pas )
  (Function: demo application of the Splitbar CDEF 17)
  (History: 3/14/89 Original by Prototyper. )
  (Modified to work right: By Kirk Chase )

uses
  Test_Window, MyGlobals, MyScroll;

var
  myEvent: EventRecord; (Event record for all events)
  doneFlag: boolean; (Exit program flag)
  code: integer; (Determine event type)
  SavePort, whichWindow: WindowPtr;
  tempRect, GrowRect, DragRect: Rect; (Rect for dragging)
  mResult: longint; (Menu list and item selected values)
  theMenu, theItem: integer; (Menu list and item selected)
  chCode: integer; (Key code)
  ch: char; (Key pressed in Ascii)
  IBeam: CursHandle; (IBeam Cursor)
  sleep: integer; (MF sleep period)
  DoIt: boolean;
  ResumePeek: WindowPeek;
  sysResult: boolean;

procedure D_About; (puts up About Box)
var
  GetSelection: DialogPtr; (Name of dialog)
  ItemHit: integer;
  ExitDialog: boolean; (Flag used to exit the Dialog)

begin
  GetSelection := GetNewDialog(AboutDialogID, nil, Pointer(-1)); (Bring in the dialog resource)
  ShowWindow(GetSelection);
  SelectWindow(GetSelection);
  SetPort(GetSelection);
  ExitDialog := FALSE;
  repeat
    ModalDialog(nil, itemHit); (Wait until an item is hit)
    if (ItemHit = About_OK) then
      begin
        ExitDialog := TRUE;
      end;
  until ExitDialog;
  DisposDialog(GetSelection);
end; (of D_About)

procedure Init_My_Menus; (Initialize the menus)
begin
  ClearMenuBar; (Clear any old menu bars)

  AppleMenuHandle := GetMenu(AppleMenuID);
  AddResMenu(AppleMenuHandle, 'DRVR'); (Add in DAs)
  InsertMenu(AppleMenuHandle, 0);

  FileMenuHandle := GetMenu(FileMenuID);
  InsertMenu(FileMenuHandle, 0);
  EditMenuHandle := GetMenu(EditMenuID);
  InsertMenu(EditMenuHandle, 0);

  DrawMenuBar; (Draw the menu bar)
end; (of Init_My_Menus)

procedure FixMenus; (adjust menu items )
begin

```

```

if (FrontWindow <> nil) then (is there a front window?)
begin
  if (FrontWindow <> MyWindow) then (window mine?)
  begin (somebody else's window is in front)
    DisableItem(FileMenuHandle, C_New);
    DisableItem(FileMenuHandle, C_Close);
    EnableItem(EditMenuHandle, C_Cut);
    EnableItem(EditMenuHandle, C_Copy);
    EnableItem(EditMenuHandle, C_Paste);
  end
  else (my window is up in front)
  begin
    EnableItem(FileMenuHandle, C_Close);
    DisableItem(FileMenuHandle, C_New);

    DisableItem(EditMenuHandle, C_Undo);

    if (EditText^.selStart <> EditText^.selEnd)
then (is there an non empty selection?)
  begin (non empty selection)
    EnableItem(EditMenuHandle, C_Cut);
    EnableItem(EditMenuHandle, C_Copy);
  end
  else
  begin
    DisableItem(EditMenuHandle, C_Cut);
    DisableItem(EditMenuHandle, C_Copy);
  end;

  if (TEGetScrapLen <> 0) then (some TE Scrap?)
  EnableItem(EditMenuHandle, C_Paste)
  else
  DisableItem(EditMenuHandle, C_Paste);
end;
end
else
begin (no front window)
  DisableItem(EditMenuHandle, C_Cut);
  DisableItem(EditMenuHandle, C_Copy);
  DisableItem(EditMenuHandle, C_Paste);
  DisableItem(FileMenuHandle, C_Close);
  EnableItem(FileMenuHandle, C_New);
end;
end; (of FixMenus)

procedure FixCursor; (Non MF Cursor adjust)
var
  mousePt: Point;
  ContentRect: Rect;
begin
  if FrontWindow = nil then
    InitCursor
  else if FrontWindow = MyWindow then
  begin
    GetMouse(mousePt);
    ContentRect := MyWindow^.portRect;
    ContentRect.right := Split^.controlRect.left;
    ContentRect.bottom := ContentRect.bottom - SBarWidth;
    if (PtInRect(mousePt, ContentRect)) then
      SetCursor(IBeam^^)
    else
      InitCursor;
  end;
end; (of FixCursor)

procedure Handle_My_Menu (var doneFlag: boolean; theMenu,
theItem: integer); (Handle menu selections)
var
  DNA: integer;
  BoolHolder: boolean;
  DAName: Str255;
  SavePort: GrafPtr;

begin
  (Start of procedure)

```

```

case theMenu of
  AppleMenuID: (Do selected menu list)
  begin
    case theItem of (Handle all commands in menu)
      About_Splitbar:
      begin
        D_About; (Call a dialog for About)
      end;
      otherwise (Handle the DAs)
      begin (Start of Otherwise)
        GetPort(SavePort); (Save current port)
        GetItem(AppleMenuHandle, theItem, DAName);
        DNA := OpenDeskAcc(DAName);
        SetPort(SavePort);
      end;
    end;
  end;
  End of item case)
  End for this list)

FileMenuID:
begin
  case theItem of
    C_New:
    begin
      Open_Test_Window;
    end;
    C_Close:
    begin
      Close_Test_Window(MyWindow);
    end;
    C_Quit:
    begin
      doneFlag := TRUE;
    end;
  end;
  End of item case)
  End for this list)

EditMenuID:
begin
  BoolHolder := SystemEdit(theItem - 1); (Do DA )
  if not (BoolHolder) then (If not DA get it)
  begin
    case theItem of
      C_Undo:
      begin
        end;
      C_Cut:
      begin
        ScrollToSelection;
        TECut(EditText); (a Cut in a TE area)
        AdjustScrollBars;
        AdjustText(1);
        AdjustText(2);
        UpdateOtherPane;
        ScrollToSelection;
      end;
      C_Copy:
      begin
        ScrollToSelection;
        TECopy(EditText); (a Copy in a TE area)
        UpdateOtherPane;
        ScrollToSelection;
      end;
      C_Paste:
      begin
        ScrollToSelection;
        TEPaste(EditText); (a Paste in a TE area)
        AdjustScrollBars;
        AdjustText(1);
        AdjustText(2);
        UpdateOtherPane;
        ScrollToSelection;
      end;
    end;
  end;
  End of item case)
end;
end;

```

```

    end;                                (End for this list)
end;                                    (End for lists)

HiliteMenu(0);    (Turn menu selection off)
end; (of Handle_My_Menu)

procedure InitMac; (Initialize Mac Stuff)
var
    MPtr: ^integer;
begin
    MoreMasters;
    InitGraf(@thePort);
    InitFonts;
    InitWindows;
    InitMenus;
    TEInit;
    InitDialogs(nil);
    FlushEvents(everyEvent, 0);
    InitCursor;

    MPtr := pointer(mbarHeightGlobal);
    mBarHeight := MPtr^;
end; (of InitMac)

procedure InitApp; (Initialize Application Stuff)
begin
    doneFlag := FALSE; (Do not exit program yet)
    Init_My_Menus; (Initialize menu bar)
    IBeam := GetCursor(IBeamCursor); (Get IBeam Cursor)

    DragRect := screenbits.bounds; (set drag rect)
    InsetRect(DragRect, 10, 10);
    DragRect.top := DragRect.top + mBarHeight;

    theErr := SysEnviron(1, theWorld);
    if (theWorld.machineType >= 0) and
    (NGetTrapAddress(WNETrapNum, ToolTrap) =
    NGetTrapAddress(UnimplTrapNum, ToolTrap)) then
        WNE := false
    else
        WNE := true;
        sleep := 10;

    if TEFrmScrap <> noErr then (get TE Scrap from Scrap)
        TETestScrapLen(0);

    EditText := nil;    (Init EditText TE Record)
    MyWindow := nil;    (Initialize the window)
end; (of InitApp)

(=====)
begin    (of main program)
    InitMac;
    InitApp;

    Open_Test_Window;    (Open the window routines)

    repeat    (Start of main event loop)
        FixCursor;
        FixMenus;

        if (EditText <> nil) then (See if a TE is active)
            TEIdle(EditText); (Blink the cursor if everything is ok)

        if WNE then
            DoIt := WaitNextEvent(everyEvent, myEvent, sleep, nil)
        else
            begin
                SystemTask;
                DoIt := GetNextEvent(everyEvent, myEvent);
            end;

        if DoIt then (If event then...)
            begin    (Start handling the event)
                code := FindWindow(myEvent.where, whichWindow);

```

```

case myEvent.what of    (Decide type of event)
    MouseDown:    (Mouse button pressed)
        begin
            if (code = inMenuBar) then
                begin
                    mResult := MenuSelect(myEvent.Where);
                    theMenu := HiWord(mResult);
                    theItem := LoWord(mResult);
                    Handle_My_Menu(doneFlag, theMenu,
theItem); (Handle the menu)
                end; (of inMenuBar)

            if (code = InDrag) then
                begin
                    DragWindow(whichWindow, myEvent.where,
DragRect); (Drag the window)
                end; (of InDrag)

            if (code = inGrow) then
                begin    (Handle the growing)
                    EraseRect(Split^.controlRect);

                    SetRect(GrowRect, 55, 55 +
Split^.controlValue, 1000, 1000);
                    mResult := GrowWindow(whichWindow,
myEvent.where, GrowRect); (Grow it)
                    SizeWindow(whichWindow, LoWord(mResult),
HiWord(mResult), TRUE); (Resize to result)
                    FixScrollbarRects;
                    DrawGrowIcon(whichWindow);
                    AdjustScrollbar;
                    DrawControls(MyWindow);
                    AdjustText(1);
                    AdjustText(2);
                    ScrollToSelection;
                end; (of doing the growing)

            if (code = inGoAway) then
                begin
                    if TrackGoAway(whichWindow, myEvent.where)
then (See if mouse released in GoAway box)
                        begin
                            Close_Test_Window(MyWindow);
                        end;
                    end; (of InGoAway)

            if (code = inContent) then
                begin
                    if (whichWindow <> FrontWindow) then
                        SelectWindow(whichWindow)
                    else
                        begin
                            SetPort(whichWindow);
                            Do_Test_Window(myEvent);
                        end;
                    end; (End of inContent)

            if (code = inSysWindow) then
                SystemClick(myEvent, whichWindow);
            end; (of MouseDown)

    KeyDown, AutoKey: (Handle key inputs)
        begin
            with myevent do
                begin
                    chCode := BitAnd(message, CharCodeMask);
                    ch := CHR(chCode);    (Change to ASCII)
                    if (Odd(modifiers div CmdKey)) then
                        begin
                            mResult := MenuKey(ch);
                            theMenu := HiWord(mResult);
                            theItem := LoWord(mResult);
                            if (theMenu <> 0) then

```



```

        Handle_My_Menu(doneFlag, theMenu,
theItem); (Do the menu selection)

        end
    else if (EditText <> nil) then
    begin
        TEDeactivate(EditText);
        TEKeyCch, EditText);
        TEActivate(EditText);
        ScrollToSelection;
        UpdateOtherPane;
    end;
    end; (End for with)
end; (of KeyDown,AutoKey)

UpDateEvt: (Update event for a window)
begin
    whichWindow := WindowPtr(myEvent.message);
    if whichWindow = MyWindow then
    begin
        GetPort(SavePort); (Save current port)
        SetPort(MyWindow); (Set port to my window)
        BeginUpdate(whichWindow);
        Update_Test_Window; (Update this window)
        EndUpdate(whichWindow);
        SetPort(SavePort);
    end;
    end; (of UpDateEvt)

ActivateEvt: (Window activated event)
begin (Handle the activation)
    whichWindow := WindowPtr(myevent.message);
    if odd(myEvent.modifiers) then
        SelectWindow(whichWindow);
    end; (of ActivateEvt)

MultiEvt:
begin
    if Odd(myEvent.message) then
    begin (resume event)
        if FrontWindow = MyWindow then
        begin
            SetPort(MyWindow);
            InvalRect(MyWindow^.portRect);
            FixMenus;
        end
        else if FrontWindow <> nil then
        begin
            ResumePeek := WindowPeek(FrontWindow);
            if ResumePeek^.windowKind < 0 then
            begin
                myEvent.what := activateEvt;
                BitSet(@myEvent.modifiers, bit0);
                sysResult :=
SystemEvent(myEvent);

                end;
            end;
            end (of resume event)

        else (suspend event)
        begin
            if FrontWindow = MyWindow then
            begin
                SetPort(MyWindow);
                InvalRect(MyWindow^.portRect);
                EnableItem(EditMenuHandle, C_Undo);
                EnableItem(EditMenuHandle, C_Cut);
                EnableItem(EditMenuHandle, C_Copy);
                EnableItem(EditMenuHandle, C_Paste);
                DrawGrowIcon(MyWindow);
            end
            else if FrontWindow <> nil then
            begin
                ResumePeek :=

```

```

WindowPeek(FrontWindow);
            if ResumePeek^.windowKind < 0 then
            begin
                myEvent.what := activateEvt;
                BitClr(@myEvent.modifiers, bit0);
                sysResult :=
SystemEvent(myEvent);

                end;
            end;
            end; (of MultiEvt)

        end; (End of case)

    end; (end of GetNextEvent)
    until doneFlag; (End of the event loop)

end. (End of the program)

```

Listing: SplitbarTest.r

```

*****
* RMaker resource file sources.
* File: SplitbarTest.R
*****

SplitbarTest.RSRC
????????

INCLUDE XP= 40:LSP 2.0:splitbar f:Splitbar Icon
INCLUDE XP= 40:LSP 2.0:splitbar f:Splitbar.CDEF 17

Type KCSB = STR
,0
© 1989 by Kirk Chase and MacTutor\00Version 1.0 March 1989

Type SIZE = GNRL
, -1
.H
4800
.L
128000
.L
80000
.I

Type BNDL
, 128
KCSB 0
ICN#
0 128
FREF
0 128

Type FREF
, 128
APPL 0

Type DLOG
*

, 1
About
50 120 189 380
Visible NoGoAway
1
1
1

* This is the DIALOG or ALERT item list.
*
Type DITL
*

```

,1
3

Button Enabled
86 83 112 163
OK

StaticText
51 24 68 241
© 1989 Kirk Chase and MacTutor

StaticText
26 37 43 218
Splitbar Test by Kirk Chase

Type WIND
*

,2
Test Window
41 3 338 508
InVisible GoAway
0
2

Type CNTL
*

,10
SplitBar
0 490 282 506
Visible
274
10
0 276 0

,9
Scrollbar2

0 490 0 506
Visible
16
9
1 100 1

,8
Scrollbar1
6 490 282 506
Visible
16
8
1 100 1

Type MENU
*

,1
\14
About Splitbar... ;;APPLE menu title
(-

,2
File
(New
(-
Close
(-
Quit/Q

,3
Edit
(Undo/Z
(-
Cut/X
Copy/C
Paste/V



Programmer's Workshop

Spray Can and Paint Bucket Tools



Kevin Parichan
Reedley, CA
Volume 5, Number 6

[Kevin Parichan is a senior, computer science major attending Cal Poly in San Luis Obispo. He has been programming on the Mac since early 1985 when he got his first computer, the 128K Mac. Right now he is doing some work using 4th Dimension.]

Painting Tools

When the Macintosh was first introduced, along with it came MacPaint and all those nifty painting tools that first awed us.

I'm the kind of person who likes to take things apart, see how they work, and see if I can figure them out. So it has been with MacPaint's tools.

We can all certainly create cheap imitation tools that expand ovals and rectangles that flash every time we move them. And since there are other, and far superior, painting programs around these days, certainly other people have figured out the mysteries of MacPaint. But I wanted to figure them out for myself.

I thought what better way than to create a Pascal unit that can be used by anybody whose program's purpose does not emphasize graphics, but would still like to supply the standard tools to their users.

Since completing this unit (right now only in a Turbo Pascal version, and available on GENie) I thought that I might share some of my knowledge with the rest of the people out there who are still scratching their heads.

Of course, I won't show you everything. That would be too easy. I'll just show you how you can create both the Spray Can and Paint Bucket tools using some simple logic and those new Quickdraw routines that surfaced with the MacPlus.

Along the way I did get some help

Technical Note #163 gives a very short code fragment on how to do a color fill routine. The problem is that SeedCFill and SeedFill want different parameters. Plus, SeedCFill uses Color Quickdraw which of course doesn't much help those of us who can't afford a Mac II.

You may also notice the function, NewBitMap. I've been using this since I first saw it in Scott Boyd's article on animated bitmaps. It's a handy little function.

I've tried to set up the routines as independent from the program as possible and to do this I use the current grafport to pass information to the routines. For both tools we use the portBits, portRect, and pnPat fields.

SeedFill and CopyMask were added when the MacPlus was introduced. They are both described in Inside Mac Vol. IV. Both are routines that operate on a portion of a bitmap, but I'll be using them on whole bitmaps which has yet to cause any problems. The spray can tool uses only CopyMask, so let's start with that.

The Spray Can

```
PROCEDURE CopyMask (srcBits, maskBits, dstBits: BitMap;  
srcRect, maskRect, dstRect: Rect);
```

As Inside Mac states: "... it transfers a bit image from the source bitmap to the destination bitmap only where the corresponding bit of the mask rectangle is a 1." As you can see in the example below.

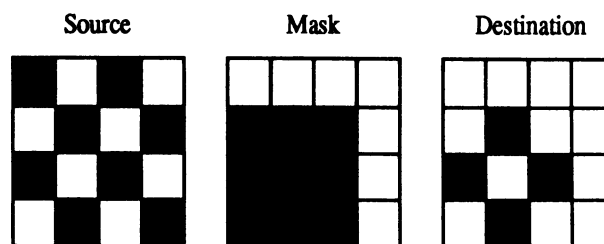


Figure 1

So for the spray can tool what we're going to do is create a source bitmap that contains the pattern we want to spray with, and a mask bitmap which has 1's corresponding to where we want to paint. In our case, a spray pattern.

In both examples, it's the call to CopyMask which is used to apply a pattern.

First thing we need to do is get the information we need from the current grafport.

```
GetPort(workPort);  
workBits:= workPort^.portBits;  
workRect:= workPort^.portRect;  
workPat:= workPort^.pnPat;
```

Then set up the mask bitmap representing an exact duplicate of the spray can cursor, and also the source bitmap which is filled with the pattern we want to spray with.

You may notice that the source bitmap is the same size as the area we are spraying into. That is because we want a continuous pattern throughout the window. If we used a smaller area, then the pattern from one spraying to the next would overlap and give us garbage on the screen. Also, the bounds for SprayBits is 17 pixels on edge, while a cursor is 16 pixels on edge. That has to do with the way QuickDraw defines rectangles. I don't understand it fully, but I do know that changing the 16's into 15's bombs the program when you use the spray can.

As everybody does when they get a new piece of code: experiment with it.

```
GetIndString(theStr, 128, 1);
```

```

SetRect(tempRect,0,0,16,16);
if NewBitMap(SprayBits,tempRect) = nil then Exit;
StuffHex(SprayBits.baseAddr,theStr);
if NewBitMap(PatBits,workRect) = nil then begin
  DisposPtr(SprayBits.baseAddr);
  Exit;
end;
SetPortBits(PatBits);
FillRect(workRect,workPat);
SetPortBits(workBits);

```

Next is the main loop. The tickcount is used to slow down the motion so that too much isn't sprayed at one time. Slowing down too much makes the spraying drag behind your mouse movements.

```

repeat
  GetMouse(where);
  with where do
    SetRect(tempRect,h-8,v-8,h+8,v+8);
    tickValue:= TickCount + 1;
    repeat until (tickValue <= TickCount);
    CopyMask(PatBits, SprayBits, workBits, tempRect,
    SprayBits.bounds, tempRect);
    tickValue:= TickCount;
    repeat until (tickValue <= TickCount);
until NOT Button;

```

That's all there is to the spray can, so now onto the...

The Paint Bucket

For the paint bucket we also need to use SeedFill.

```

PROCEDURE SeedFill (srcPtr: Ptr; srcRow, dstRow,
height, words, seedH, seedV: Integer);

```

As Inside Mac state, "... computes a destination bit image with 1's only in the pixels where paint can leak from the starting seed point ..." All we need to do then after calling SeedFill is apply a pattern to the destination bit image which can be accomplished with CopyMask

SeedFill uses pointers pointing into bitmaps. It's easier just to copy the working image to a new bitmap and set the pointers to its baseAddr and it helps in other ways as explained below. Besides, windows usually only show a view of a much larger image. If SeedFill is to work on a whole image you would use the larger image's baseAddr anyway. But if you really know what you're doing and only want to fill a portion of the bitmap, then I'm afraid you're going to have to do some pointer arithmetic.

Get the same information as before.

```

GetPort(workPort);
workBits:= workPort^.portBits;
workRect:= workPort^.portRect;
workPat:= workPort^.pnPat;

```

Set up our two bitmaps for the calls to SeedFill and CopyMask. Make a copy of the image we are using the paint bucket on, which gets our addresses right and also helps us because the current grafport is a window and its bitmap also contains the title bar. You can also see where this code needs modification. What about scroll bars and the grow icon. A

simple fix is to pass to the routines the area you want to fill or spray in. Don't think clipping the area will work. SeedFill is not clipped to the current grafport. Take out the code that makes a copy of the work image and see for yourself.

```

if NewBitMap(dstMap,workRect) = nil then Exit;
if NewBitMap(srcMap,workRect) = nil then begin
  DisposPtr(dstMap.baseAddr);
  Exit;
end;
CopyBits(workBits,srcMap,workRect,workRect,srcCopy,nil);

```

Now we check to see if we are filling on a black pixel. To act just like a normal paint bucket we need to apply the pattern to the black areas instead of the white. But SeedFill only works on white, so we invert the image before and after calling SeedFill.

```

onBlack:= GetPixel(where.h,where.v);
if onBlack then begin
  SetPortBits(srcMap);
  InvertRect(workRect);
  SetPortBits(workBits);
end;

```

Set up the bitmap containing the pattern we want to fill with. Don't forget to invert the pattern as well if we are starting the fill on a black pixel.

```

if NewBitMap(PatBits,workRect) = nil then begin
  DisposPtr(dstMap.baseAddr);
  DisposPtr(srcMap.baseAddr);
  Exit;
end;
SetPortBits(PatBits);
FillRect(workRect,workPat);
if onBlack then InvertRect(workRect);
SetPortBits(workBits);

```

This is the area that I don't want to be bothered with. I just say that I'm attempting to fill the entire bitmap. Don't forget to make words an even number. Height is the height of our image in pixels and words is the width of our image in words (as in 2 bytes).

```

srcPtr:= srcMap.baseAddr;
srcRow:= srcMap.rowBytes;
dstPtr:= dstMap.baseAddr;
dstRow:= dstMap.rowBytes;
height:= dstMap.bounds.bottom - dstMap.bounds.top;
words:= (dstRow + 1) DIV 2;

```

And here it is. The code that does all the real work. Simple, isn't it. Where is the point where the mouse was clicked in the window in local coordinates.

```

SeedFill(srcPtr,dstPtr,srcRow,dstRow,height,words,where.h,where.v);
CopyMask(PatBits,dstMap,srcMap,workRect,workRect,srcMap.bounds);

```

We check to see if we started filling on a black pixel, and if so, invert our image once again, then copybit it to the grafport.

```

if onBlack then begin
  SetPortBits(srcMap);

```

```

    InvertRect(workRect);
    SetPortBits(workBits);
end;
CopyBits(srcMap,workBits,workRect,workRect,srcCopy,nil);

```

Well, that's it. Not much to it. There are some obvious places to improve upon, but like I said, I tried to make the code as generic as possible. It's your job to customize it to suit your needs.

Listing: ToolDemo.pas

Program ToolDemo;

```

($U-)
($R ToolDemoRes)

```

Uses

```
MemTypes, QuickDraw, OSIntf, ToolIntf;
```

Const

```

AppleMenu    = 128;
FileMenu     = 129;
EditMenu     = 130;
ToolMenu     = 131;
  SprayItem   = 1;
  BucketItem  = 2;
PatternMenu  = 132;
  WhiteItem   = 1;
  LtGrayItem  = 2;
  GrayItem    = 3;
  DkGrayItem  = 4;
  BlackItem   = 5;

```

Var

```

myMenus      : array [AppleMenu..PatternMenu] of MenuHandle;
myWindow     : WindowPtr;
Finished     : Boolean;
GrowArea     : Rect;
CurrentPat   : Integer;
CurrentTool  : Integer;

```

```

(*****

```

Function NewBitMap (VAR theBitMap : BitMap; theRect : Rect) : Ptr;

Begin

```

NewBitMap:= nil;
with theBitMap, theRect do begin
  rowBytes:= ((right-left+15) DIV 16) *2;
  baseAddr:= NewPtr(rowBytes * (bottom-top));
  bounds:= theRect;
  if MemError = noErr then
    NewBitMap:= baseAddr;
end;

```

End;

End;

Procedure DoSprayCan (where : Point);

Var

```

workPort    : GrafPtr;
workBits    : BitMap;
workRect    : Rect;
workPat     : Pattern;
theStr      : Str255;
tempRect    : Rect;
SprayBits   : BitMap;
PatBits     : BitMap;
tickValue   : LongInt;

```

Begin

```

GetPort(workPort);
workBits:= workPort^.portBits;
workRect:= workPort^.portRect;
workPat:= workPort^.pnPat;

```

```
GetIndString(theStr, 128, 1);
```

```

SetRect(tempRect, 0, 0, 16, 16);
if NewBitMap(SprayBits, tempRect) = nil then Exit;
StuffHex(SprayBits.baseAddr, theStr);

```

```

if NewBitMap(PatBits, workRect) = nil then begin
  DisposPtr(SprayBits.baseAddr);
  Exit;
end;

```

```

SetPortBits(PatBits);
FillRect(workRect, workPat);
SetPortBits(workBits);

```

repeat

```
GetMouse(where);
```

```
with where do
```

```
SetRect(tempRect, h-8, v-8, h+8, v+8);
```

```
tickValue:= TickCount + 1;
```

```
repeat until (tickValue <= TickCount);
```

```
CopyMask(PatBits, SprayBits, workBits, tempRect, SprayBits.bounds, tempRect);
```

```
tickValue:= TickCount;
```

```
repeat until (tickValue <= TickCount);
```

```
until NOT Button;
```

```
DisposPtr(PatBits.baseAddr);
```

```
DisposPtr(SprayBits.baseAddr);
```

End;

Procedure DoPaintBucket (where : Point);

Var

```
workPort    : GrafPtr;
```

```
workBits    : BitMap;
```

```
workRect    : Rect;
```

```
workPat     : Pattern;
```

```
PatBits     : BitMap;
```

```
onBlack     : Boolean;
```

```
srcMap      : BitMap;
```

```
dstMap      : BitMap;
```

```
srcPtr      : Ptr;
```

```
dstPtr      : Ptr;
```

```
srcRow      : Integer;
```

```
dstRow      : Integer;
```

```
height      : Integer;
```

```
words       : Integer;
```

Begin

```
GetPort(workPort);
```

```
workBits:= workPort^.portBits;
```

```
workRect:= workPort^.portRect;
```

```
workPat:= workPort^.pnPat;
```

```
if NewBitMap(dstMap, workRect) = nil then Exit;
```

```
if NewBitMap(srcMap, workRect) = nil then begin
```

```
DisposPtr(dstMap.baseAddr);
```

```
Exit;
```

end;

```
CopyBits(workBits, srcMap, workRect, workRect, srcCopy, nil);
```

```
onBlack:= GetPixel(where.h, where.v);
```

```
if onBlack then begin
```

```
SetPortBits(srcMap);
```

```
InvertRect(workRect);
```

```
SetPortBits(workBits);
```

end;

```
if NewBitMap(PatBits, workRect) = nil then begin
```

```
DisposPtr(dstMap.baseAddr);
```

```
DisposPtr(srcMap.baseAddr);
```

```
Exit;
```

end;

```
SetPortBits(PatBits);
```

```
FillRect(workRect, workPat);
```

```
if onBlack then InvertRect(workRect);
```

```
SetPortBits(workBits);
```

```
srcPtr:= srcMap.baseAddr;
```

```

srcRow:= srcMap.rowBytes;
dstPtr:= dstMap.baseAddr;
dstRow:= dstMap.rowBytes;
height:= dstMap.bounds.bottom - dstMap.bounds.top;
words:= (dstRow + 1) DIV 2;

SeedFill(srcPtr,dstPtr,srcRow,dstRow,height,words,where.h,where.v);
CopyMask(PatBits,dstMap,srcMap,workRect,workRect,srcMap.bounds);

if onBlack then begin
  SetPortBits(srcMap);
  InvertRect(workRect);
  SetPortBits(workBits);
end;
CopyBits(srcMap,workBits,workRect,workRect,srcCopy,nil);

DisposPtr(srcMap.baseAddr);
DisposPtr(dstMap.baseAddr);
DisposPtr(PatBits.baseAddr);
End;

(*****
Procedure ProcessMenu (codeWord : LongInt);
Var
  menuNum : Integer;
  itemNum : Integer;
  itemStr : Str255;
  dummy : Integer;
Begin
  if codeWord <> 0 then begin
    menuNum := HiWord(codeWord);
    itemNum := LoWord(codeWord);
    case menuNum of
      AppleMenu :
        begin
          GetItem(myMenus[AppleMenu],itemNum,itemStr);
          dummy:= OpenDeskAcc(itemStr);
        end;
      FileMenu : Finished:= TRUE;
      EditMenu : if NOT SystemEdit(itemNum - 1) then ;
      ToolMenu :
        begin
          CheckItem(myMenus[ToolMenu],CurrentTool,false);
          CurrentTool:= itemNum;
          CheckItem(myMenus[ToolMenu],CurrentTool,true);
        end;
      PatternMenu :
        begin
          CheckItem(myMenus[PatternMenu],CurrentPat,false);
          CurrentPat:= itemNum;
          CheckItem(myMenus[PatternMenu],CurrentPat,true);
          SetPort(myWindow);
          case CurrentPat of
            WhiteItem : PenPat(white);
            LtGrayItem : PenPat(ltGray);
            GrayItem : PenPat(gray);
            DkGrayItem : PenPat(dkGray);
            BlackItem : PenPat(black);
          end
        end;
      end; (case)
      HiliteMenu(0);
    end; (big if)
  end;
End;

(*****
Procedure DealWithMouseDown(theEvent: EventRecord);
Var
  whichWindow : WindowPtr;
  mouseLoc : Point;
  windowLoc : Integer;
  position : LongInt;
Begin
  mouseLoc:= theEvent.where;

```

```

  windowLoc:= FindWindow(mouseLoc,whichWindow);
  case windowLoc of
    inMenuBar : ProcessMenu(MenuSelect(mouseLoc));
    inSysWindow : SystemClick(theEvent,whichWindow);
    inDrag :
      DragWindow(whichWindow,mouseLoc,screenBits.bounds);
    inGoAway : if TrackGoAway(whichWindow,mouseLoc) then
      Finished:= true;
    inGrow :
      begin
        position:= GrowWindow(whichWindow,mouseLoc,GrowArea);
        if position <> 0 then begin
          SizeWindow(whichWindow,loword(position),hiword(position),false);
          SetPort(whichWindow);
          InvalRect(whichWindow^.portRect);
        end;
      end;
    inZoomIn,inZoomOut :
      begin
        if TrackBox(whichWindow,mouseLoc>windowLoc) then
          begin
            SetPort(whichWindow);
            ClipRect(whichWindow^.portRect);
            EraseRect(whichWindow^.portRect);
            ZoomWindow(whichWindow>windowLoc,true);
            InvalRect(whichWindow^.portRect);
          end;
        end;
        inContent :
          begin
            if whichWindow <> FrontWindow then
              SelectWindow(whichWindow)
            else begin
              SetPort(whichWindow);
              GlobalToLocal(mouseLoc);
              case CurrentTool of
                SprayItem : DoSprayCan(mouseLoc);
                BucketItem : DoPaintBucket(mouseLoc);
              end;
            end;
          end;
        end;
      end;
  end;
End;

Procedure DealWithKeyDowns(theEvent: EventRecord);
Var
  CharCode : char;
Begin
  CharCode:= CHR(BitAnd(theEvent.message,charCodeMask));
  if BitAnd(theEvent.modifiers,CmdKey) = CmdKey
  then ProcessMenu(MenuKey(CharCode));
End;

Procedure DealWithActivates(theEvent: EventRecord);
Var
  theWindow: WindowPtr;
Begin
  theWindow := WindowPtr(theEvent.message);
  if Odd(theEvent.modifiers)
  then SetPort(theWindow);
End;

Procedure DealWithUpdates(theEvent: EventRecord);
Var
  theWindow : WindowPtr;
  tempPort : WindowPtr;
Begin
  theWindow := WindowPtr(theEvent.message);
  GetPort(tempPort);
  SetPort(theWindow);
  BeginUpDate(theWindow);
  ClipRect(theWindow^.portRect);
  EraseRect(theWindow^.portRect);
  PenSize(5,5);

```

```

    FrameOval(theWindow^.portRect);
    PenSize(1,1);
    EndUpdate(theWindow);
    SetPort(tempPort);
End;

Procedure MainEventLoop;
Var
    Event : EventRecord;
Begin
    repeat
        SystemTask;
        if GetNextEvent(everyEvent, Event) then
            case Event.what of
                mouseDown : DealWithMouseDown(Event);
                AutoKey : DealWithKeyDown(Event);
                KeyDown : DealWithKeyDown(Event);
                ActivateEvt : DealWithActivates(Event);
                UpdateEvt : DealWithUpdates(Event);
            end; (case)
    until Finished;
End;

(*****)
Procedure SetupStuff;
Var
    index : Integer;
Begin
    MaxApplZone;
    InitGraf(@thePort);
    InitFonts;
    InitWindows;
    InitMenus;
    TEInit;
    InitDialogs(nil);

    for index:= AppleMenu to PatternMenu do begin
        myMenus[index] := GetMenu(index);
        InsertMenu(myMenus[index],0);
    end;
    AddResMenu(myMenus[AppleMenu], 'DRVr');
    DrawMenuBar;
    CurrentTool:= SprayItem;
    CurrentPat:= BlackItem;
    CheckItem(myMenus[ToolMenu],CurrentTool,true);
    CheckItem(myMenus[PatternMenu],CurrentPat,true);
    myWindow:= GetNewWindow(1000,nil,pointer(-1));

    Finished:= false;
    with screenBits.bounds do
        SetRect(GrowArea, 150, 150, right, bottom);

    FlushEvents(everyEvent,0);

```

```

    InitCursor;
End;

Begin
    SetupStuff;
    MainEventLoop;
End.

```

Listing: ToolDemo.r

```

ToolDemoRes

Type MENU
, 128
\14

, 129
File
Quit/Q

, 130
Edit
Undo/Z
(-
Cut/X
Copy/C
Paste/V
Clear

, 131
Tools
Spray Can
Paint Bucket

, 132
Patterns
White/1
Light Gray/2
Gray/3
Dark Gray/4
Black/5

Type STR#
, 128
1
0100001020800400002282001048010104409008022208004120000408000080

Type WIND
, 1000
Tool Demo
50 50 300 400
Visible GoAway
8
0

```



Advanced Mac'ing

List of Controls LDEF



Pascal

James Plamondon
Berkeley, CA
Volume 5, Number 6

List-Of-Controls LDEF Introduction

I recently needed to implement a list, in which each cell would be an active control. It turned out to be a little harder to do than I thought it would be. To save others the trouble, I thought I'd describe how I made it all work.

Overview

I use Apple's List Manager to manipulate my list of controls. Each element of the list contains a ControlHandle to a PopUp-Menu control. Whenever a mouseDown event occurs in the list, I find out if it is one of the controls, and if so, call TrackControl() (IM v1 p323) to deal with it. If the mouseDown is in the list, but not in an active part of the control, I pass the event off to LClick() (IM v4 p273), so the Apple List Manager can handle it.

Whenever the user selects or unselects a list element, or brings a new control into view by scrolling the list, the List Manager will send a message to the list's LDEF telling it to draw or hilite the list element. My LDEF will then set the control's hilite state as needed to assure that it is drawn properly, and then call Draw1Control() (IM v4 p53) to do the actual drawing.

A routine called FixCtlRects() is called in the test program whenever a control may have changed position in the list's display, to make sure that its controlRect field matches its list rectangle, and that it won't be drawn if it is out of the list's display area.

The example program uses the PopUp-Menu control described in MacTutor, September 1988, PopUp-Menu Control CDEF. I could have used a different CDEF, but I would have had to include the CDEF's source code, and I didn't want to make this article any longer than necessary.

LDEFs

LDEFs are described on pages 278-277 of Inside Mac, Volume Four, in the List Manager chapter (pages 259-282). The 'LDEF' associated with a list is loaded into memory when the list is created. Like other _DEFs (MDEFs, CDEFs, etc.), the code resource has a single entry point at its first byte (i.e., it has no jump table). The entry point must be a procedure with the following definition:

```
PROCEDURE MyLDEF(  
  1Message: INTEGER;  
  1Select: Boolean;  
  1Rect: Rect;  
  1Cell: Cell;  
  1DataOffset,  
  1DataLen: INTEGER;  
  1Handle: ListHandle);
```

The function and argument names may be changed, of course, but their order and type must be as shown here (from IM v4 p276).

When a new list is created using the LNew() function (IM v4 p270), one of the parameters you must give to LNew() is the resource ID of the LDEF you want the list to use. Apple's default LDEF has a resource ID of 0. Our LDEF has a resource ID of RES_ID (128). When we create the new list, the List Manager will load the specified LDEF, and store a handle to it in the listDefProc field of the resulting list record. Whenever a cell of the list needs to be drawn or hilited, the List Manager will send a message to the LDEF, and the LDEF will do the work.

LDEF Messages

An LDEF message is just a procedure call to the LDEF's MyLDEF() routine, with 1Message (an integer) equal to a meaningful value. If the List Manager wants the LDEF to draw a cell, it calls MyLDEF() with 1Message equal to IDDrawMsg (1). If the cell is to be hilited, MyLDEF() is called with 1Message equal to IHiliteMsg (2). MyLDEF() contains a CASE statement, keyed to the 1Message parameter's value. If 1Message equals IDDrawMsg, the routine doDrawMsg() is called. If 1Message equals IHiliteMsg, doHiliteMsg() is called. That's all there is to LDEF message passing.

There are four different messages an LDEF may receive. Two of them, IDDrawMsg and IHiliteMsg, must be handled by all LDEFs.

When the LDEF gets a IDDrawMsg message, it needs to draw the given list cell. If the list cell is selected (i.e., if 1Select is TRUE), then you need to hilite the cell, to tell the user that it's been selected. The LDEF used in this article simply calls Draw1Control() to draw the control stored in the given cell. First it has to diddle with the control's rectangle, though, as we'll see later.

The IHiliteMsg message is sent to the LDEF when a previously drawn cell needs to be hilited or unhilited. In most cases, this is done by simply inverting the cell's rectangle. Our LDEF has to get a little fancier.

The other two LDEF messages, IInitMsg and ICloseMsg, need only be implemented by the LDEF if each list handled by the LDEF needs to be initialized in some way, and then have its initialization voided when the list is disposed of. In this LDEF, however, no such initialization or deinitialization need be done, so it does nothing when it receives those messages.

List Hilighting

We need to be able to draw the control in such a way as to make it look selected in the list. The mechanism for inverting the

control's title already exists in the CDEF, so we'll use that to indicate that the control is selected in the list.

The careful readers of the PopUp-Menu CDEF article and code may have noticed that a partCode was defined for the title of the pop-up menu (titlePart = 2), but that this partCode was never returned by doTestCntl(). The part code exists solely for this article. By setting the contrlHilite field of the control to be drawn to titlePart before calling Draw1Control(), we can guarantee that the control will have its title highlighted when it is drawn.

The Example Program

The example program is a very stripped Mac application. It simply initializes all of the usual managers, throws up a dialog, and cycles through ModalDialog() (IM v1 p415) until the user selects the dialog's OK button. Most of the work of handling the list and the controls it contains is done by the filterProc MyFilter() — the address of which is passed to ModalDialog() in the main program — and its helper functions, doMouseDown() and doKeyDown().

After the user clicks the OK button, the program exits the ModalDialog() loop, disposes of the controls, list, and dialog, and quits. The final values of the controls are not accessed, because I'm not actually doing anything with them. In a real program you'd get the final values out of the controls, using GetCtlMax() and GetCtlMin() as described in PopUp-Menu Control CDEF.

Using the List-Of-Controls LDEF

The LDEF expects to find a control handle in each of its cells. We have to put these handles there ourselves. First we need to create the list itself, which we do with a call to LNew() in the routine InitList(). The only interesting part of this call is the use of RES_ID (128, the resource ID of our LDEF), rather than zero (0), Apple's default LDEF.

Next, we need to create the controls we wish to add to the list. This is accomplished with calls to NewControl() (IM v1 p319), in the routine ReadData(). I read the resource ID's of the controls I need from the 'INT#' resource I've placed in the resource fork, and then read in the associated 'CNTL' resources.

I give the control a rectangle that is out of sight. It gets drawn despite my saying that I want it to be invisible (very irritating), so I just hide it, so that it can't be seen until I want it to be seen.

After getting a control handle from NewControl(), the control is made visible (but not drawn) by placing TRUE (255) in its contrlVis field. ShowControl() (IM v1 p322) would try to draw the control, but that's a waste of time, because the window is still invisible.

Finally, the control handle is placed in a new cell in the list. From there, we can extract it, and pass it along to Draw1Control() and TrackControl() as necessary.

Problems

At first I thought that I could just update a control's rectangle whenever the cell it was in received an IDrawMsg or an IHiliteMsg. But that alone won't work, because the List Manager uses ScrollRect() (IM v1 p187) to scroll the bits inside the list's

rectangle, and only redraws cells as they come into view. This means that after the initial drawing of the list, the only place in which cells are redrawn is in the top cell frame or in the bottom cell frame. If you modify the control rectangle to match the cell rectangle only when the cell is redrawn, then sooner or later, all of the controls are going to have either the top or bottom cell frame as their rectangles. It may look like there are controls in the middle of the list, but those are just scrolled bit-images; the controls will think they are in the top and bottom frames.

So, when you click in a control in the middle of the list, the control does not respond... because it's not really there. The thing you thought was a control was just a picture of a control. The control is in either the top or bottom cell frame (depending on where it was drawn last). If you click in the top or bottom cell frame, you'll click in a control, all right, but not necessarily the one you see there. Your click will get picked up by the first control in the window's control list (not the List Manager list) that was recently redrawn in that frame. Argh!

Here comes the kludge. The only way the user can move the controls on the list is with the list's scroll bar. Therefore, whenever we receive a mouseDown event in the list that is not in one of our controls, we have to call FixCtlRects(). FixCtlRects() just walks through the list, getting each cell's rectangle and setting the rectangle of the control in the cell to match. If the cell is invisible, then control is made invisible, and vice versa. This doesn't take too long for a short list, but if you had a couple of hundred controls, it would be noticeably slow.

We also have to call FixCtlRects() after reading in the controls the first time, to make sure they're displayed properly.

It's a kludge, but it works, so what the hey.

Miscellaneous Notes

There are few other things about the LDEF, the pop-up menu CDEF, and its example program that are worth mentioning.

Note that I pass POINTER(-1) to TrackControl() in doMouseDown(). This enables the pop-up menu's CDEF's autoTrack function. This is the function that pops up the PopUp-Menu via PopUpMenuSelect() (IM v5 p241), as described in detail in PopUp-Menu Control CDEF.

CenterWindow() just centers a window in the screen, about two-thirds of the way up.

Acknowledgements

I started this project in order to do something we needed to do at work, but it kind of took on a life of its own when I realized that there was an article in it. I'd like to thank the guys at Abacus — Dan, Jim, and Will — for letting me spend the time I did on this. It won't happen again... for another week or two, anyway.

Listing: Test.make

```
# Test.make
#
#
Test  ffTest.r ControlLDEF.LDEF PopMenuCDEF.CDEF
Rez   -rd Test.r -o Test
Test  ffTest.p.o          o
```

```

ControlLDEF.LDEF          d
PopupMenuCDEF.CDEF        d
Test.r
Link                      d
Test.p.o                  d
  "(Libraries)"Runtime.o    d
  "(Libraries)"Interface.o  d
  "(PLibraries)"PasLib.o    d
  -o Test

ControlLDEF.LDEF ff ControlLDEF.p.o
Link -sg ControlLDEF        d
  -rt LDEF=1                d
  -m MYLDEF ControlLDEF.p.o  d
  "(Libraries)"Interface.o  d
  -o ControlLDEF.LDEF

PopupMenuCDEF.CDEF ff PopMenuCDEF.p.o
Link -sg PopMenuCDEF        d
  -rt CDEF=1                d
  -m MYCONTROL PopMenuCDEF.p.o d
  "(Libraries)"Interface.o  d
  "(PLibraries)"PasLib.o    d
  -o PopMenuCDEF.CDEF

PopupMenuCDEF.p.o f PopMenuCDEF.p PopMenuIntf.p
Pascal PopMenuCDEF.p -o PopMenuCDEF.p.o

Test.p.o f Test.p
Pascal Test.p -o Test.p.o

ControlLDEF.p.o f ControlLDEF.p PopMenuIntf.p
Pascal ControlLDEF.p -o ControlLDEF.p.o

# END OF FILE: Test.make

```

Listing: ControlLDEF.P

```

(*****
ControlLDEF.p: List Definition Function for list of controls.
*****)

```

UNIT ControlLDEF;

INTERFACE USES

```

($U MemTypes.p      ) MemTypes,
($U QuickDraw.p     ) QuickDraw,
($U OSIntf.p        ) OSIntf,
($U ToolIntf.p      ) ToolIntf,
($U PackIntf.p      ) PackIntf,
($U PopMenuIntf.p   ) PopMenuIntf;

```

```

PROCEDURE MyLDEF( lMessage: INTEGER;lSelect: Boolean;
lRect: Rect;lCell: Cell;lDataOffset,lDataLen: INTEGER;
lHandle: ListHandle);

```

IMPLEMENTATION

CONST

```

isVIS   = 255;
notVIS  = 0;

```

TYPE

```

StateStuff = record
  oldPen: PenState;
  oldPort: GrafPtr;
  oldClip: RgnHandle;
  newClip: RgnHandle;
end; ( StateStuff )

```

```

PROCEDURE DrawCell(lSelect: Boolean;lRect: Rect;
lCell: Cell;lHandle: ListHandle);FORWARD;

```

```

(*****

```

```

MyLDEF: List Definition Function for Controls list.
Responds to LDrawMsg and lHiliteMsg; ignores
lInitMsg and lCloseMsg.
*****
)

```

```

PROCEDURE MyLDEF( lMessage: INTEGER; lSelect: Boolean;
lRect: Rect;lCell: Cell;lDataOffset,lDataLen: INTEGER;
lHandle: ListHandle);

```

BEGIN

CASE lMessage OF

```

  lInitMsg:
    ; ( no initialization needed )
  lCloseMsg:
    ; ( no deallocation needed )

```

```

  lDrawMsg, lHiliteMsg:
    DrawCell(lSelect, lRect, lCell, lHandle);

```

```

END; ( case )

```

```

END; ( MyLDEF )

```

```

(*****
SaveState: Saves the current drawing environment.
*****
)

```

```

PROCEDURE SaveState(VAR oldState: StateStuff;
lRect: Rect;lHandle: ListHandle);

```

BEGIN

WITH oldState DO BEGIN

```

  ( save the current pen state )
  GetPenState(oldPen);
  ( save current grafPort, set new grafPort )
  GetPort(oldPort);
  SetPort(lHandle^.port);
  ( allocate space for old and new clipping regions )
  oldClip := NewRgn;
  newClip := NewRgn;
  ( save old clipping region )
  GetClip(oldClip);
  ( set newClip region to given rectangle )
  RectRgn(newClip, lRect);
  ( intersection of rect and region )
  SectRgn(oldClip, newClip, newClip);
  ( set grafPorts' clip region to the intersection )
  SetClip(newClip);

```

```

END; ( with )

```

```

END; ( SaveState )

```

```

(*****
RestoreState: Restores current drawing environment.
*****
)

```

```

PROCEDURE RestoreState(oldState: StateStuff);
BEGIN

```

WITH oldState DO BEGIN

```

  ( restore the previous clipping region )
  SetClip(oldClip);
  ( restore the previous pen state )
  SetPenState(oldPen);
  ( restore the previous grafPort )
  SetPort(oldPort);
  ( dispose of the regions' storage )
  DisposeRgn(oldClip);
  DisposeRgn(newClip);

```

```

END; ( with )

```

```

END; ( RestoreState )

```

```

(*****
DrawCell: Draws the given cell, either selected or
normal, by drawing the control stored in the cell.
*****
)

```

```

PROCEDURE DrawCell(lSelect: Boolean;lRect: Rect;
lCell: Cell;lHandle: ListHandle);

```

```

VAR

```

```

oldState: StateStuff;
ch: ControlHandle;
dl: INTEGER;

BEGIN
  ( save the current drawing environment )
  SaveState(oldState, lRect, lHandle);

  ( get the cell's data )
  dl := sizeof(Handle);
  LGetCell(@ch, dl, lCell, lHandle);

  ( update the control's fields )
  with ch^^ do begin
    contrlRect := lRect;
    contrlVis := isVIS;

    ( ensure proper hiliting )
    if (lSelect) then
      contrlHilite := titlePart
    else begin
      contrlHilite := 0;
    end; ( if )
  end; ( with ch^^ )

  ( draw the control )
  DrawlControl(ch);

  ( restore the previous drawing state )
  RestoreState(oldState);
END; ( DrawCell )

END. ( ControlLDEF )

```

Listing: Test.p

```

(*****
Test.p
List-of-Controls example, using pop-up menu controls.
*****)

```

PROGRAM Test;

USES

```

($U MemTypes.p      ) MemTypes,
($U QuickDraw.p     ) QuickDraw,
($U OSIntf.p        ) OSIntf,
($U ToolIntf.p      ) ToolIntf,
($U PackIntf.p      ) PackIntf,
($U PopMenuIntf.p   ) PopMenuIntf;

```

CONST

```

RES_ID   = 128; ( resource ID's )

iOK      = 1; ( OK button )
iLIST    = 2; ( list of controls )

( visibility )
isVIS    = 255;
notVIS   = 0;

```

TYPE

```

( Pascal equivalent of 'INT#' resource type )
IntList = record
  count: INTEGER;
  int: array[1..1024] of INTEGER;
end; ( IntList )

```

```

ILPointer = ^IntList;
ILHandle = ^ILPointer;

```

VAR

```

myDialog: DialogPtr; ( test dialog )
lh: ListHandle; ( handle to list )

```

```

itemHit: INTEGER; ( user's choice )
savePort: GrafPtr; ( to save port in )

```

```

(*****
CenterWindow: Centers the given window in the screen. If the
'show' argument is TRUE, then the window is shown.
*****)

```

PROCEDURE CenterWindow(wPtr: WindowPtr; show: Boolean);

VAR

```

pRect: Rect;
wRect: Rect;
h, v: INTEGER;

```

BEGIN

```

pRect := screenBits.bounds;
wRect := WindowPeek(wPtr)^.port.portRect;
v := ((pRect.bottom - pRect.top) -
      (wRect.bottom - wRect.top)) div 3;
h := ((pRect.right - pRect.left) -
      (wRect.right - wRect.left)) div 2;
MoveWindow(wPtr, h, v, show);
if (show) then begin
  ShowWindow(wPtr);
end;

```

END; (CenterWindow)

```

(*****
FixCtlRects: Make the contrlRects of all controls in
the list match their list rectangles, and that controls
in invisible cells are also invisible.
*****)

```

PROCEDURE FixCtlRects(lh: ListHandle);

VAR

```

ch: ControlHandle;
i: INTEGER;
dl: INTEGER;
toRect: Rect;
c: Cell;

```

BEGIN

```

( default )
dl := sizeof(Handle);
( update the controls in the list )
FOR i := 0 TO (lh^.dataBounds.bottom - 1) DO BEGIN
  ( define cell to access )
  SetPt(Point(c), 0, i);
  ( get the control handle from the cell )
  LGetCell(@ch, dl, c, lh);
  IF (PtInRect(Point(c), lh^.visible)) THEN
    BEGIN
      ( get cell rect; copy into control rect )
      LRect(toRect, c, lh);
      ch^^.contrlRect := toRect;
      ( make sure the control is visible )
      ch^^.contrlVis := isVIS;
    END
  ELSE BEGIN
    ( make sure the control is invisible )
    ch^^.contrlVis := notVIS;
  END;
END;
END; ( FixCtlRects )

```

```

(*****
FindCell: Find the cell in the given list containing
the given point, which is assumed to be in local
co-ordinates. If the point is not in the list's
rectangle, the resulting cell will have both
h and v set to (-1).
*****)

```

FUNCTION FindCell(p: Point; lh: ListHandle): Cell;

```

VAR
  c: Cell;

BEGIN
  with lh^ do begin
    if (not PtInRect(p, rView)) then
      SetPt(Point(c), -1, -1)
    else with rView.topLeft do begin
      c.h := ((p.h - h) DIV cellSize.h) + visible.left;
      c.v := ((p.v - v) DIV cellSize.v) + visible.top;
    end; { else, with rView.topLeft }
  end; { with lh^ }
  FindCell := c;
END; { FindCell }

(*****
doKeyDown: Filters all keyDown events in the modal
dialog, implementing the standard keyboard
equivalents for the OK button.
*****)
FUNCTION doKeyDown(dPtr: DialogPtr; VAR theEvent:
EventRecord; VAR itemHit: INTEGER): Boolean;
VAR
  c: LONGINT;
BEGIN
  { default }
  doKeyDown := FALSE;

  { get the ascii code }
  c := BitAnd(theEvent.message, charCodeMask);

  { check for Return or Enter }
  IF ((c = 3) or (c = 13)) THEN BEGIN
    itemHit := iOK;
    doKeyDown := TRUE;
  END; { if }
END; { doKeyDown }

(*****
doMouseDown
*****)
FUNCTION doMouseDown(dPtr: DialogPtr; VAR theEvent:
EventRecord; VAR itemHit: INTEGER): Boolean;
VAR
  item: INTEGER;
  part: INTEGER;
  pt: Point;
  c: Cell;
  ch: ControlHandle;
  dbl: Boolean;
  sel: Boolean;
BEGIN
  pt := theEvent.where;
  GlobalToLocal(pt);

  { See Tech Note #112, FindDItem() }
  item := FindDItem(dPtr, pt) + 1;

  IF (item = iLIST) THEN
    BEGIN
      { in whom did the mouseDown occur? }
      part := FindControl(pt, dPtr, ch);

      IF ((ch = nil) or (ch = lh^.vScroll)) THEN
        BEGIN
          { clicked in list or in scroll bar }
          dbl := LClick(pt,
            theEvent.modifiers,
            lh);

          { fix the controls' contrlRects }
          FixCtlRects(lh);
        END
      ELSE BEGIN { clicked in pop-up menu }

```

```

        part := TrackControl(ch, pt, POINTER(-1));
        { what cell was the mouseDown in? }
        c := FindCell(pt, lh);
        { redraw the control if cell was selected }
        if (LGetSelect(false, c, lh)) then begin
          HiliteControl(ch, titlePart);
        end; { if }
      END; { else }
      doMouseDown := TRUE;
    END { item = iLIST }
  ELSE BEGIN
    doMouseDown := FALSE;
  END;
END; { doMouseDown }

```

```

(*****
doUpdateEvt
*****)

```

```

FUNCTION doUpdateEvt(dPtr: DialogPtr; VAR theEvent:
EventRecord; VAR itemHit: INTEGER): Boolean;
BEGIN
  { begin the update }
  BeginUpdate(dPtr);

  { update the dialog items }
  UpdtDialog(dPtr, dPtr^.visRgn);

  { update the list }
  LUpdate(dPtr^.visRgn, lh);

  { end the update }
  EndUpdate(dPtr);

  { always return true (we handled it) }
  doUpdateEvt := TRUE;
END; { doUpdateEvt }

```

```

(*****
MyFilter
*****)

```

```

FUNCTION MyFilter(dPtr: DialogPtr; VAR theEvent:
EventRecord; VAR itemHit: INTEGER): Boolean;
BEGIN
  CASE theEvent.what OF
    keyDown:
      MyFilter := doKeyDown(dPtr, theEvent,
        itemHit);
    mouseDown:
      MyFilter := doMouseDown(dPtr, theEvent,
        itemHit);
    updateEvt:
      MyFilter := doUpdateEvt(dPtr, theEvent,
        itemHit);
    OTHERWISE
      MyFilter := FALSE;
  END; { case }
END; { MyFilter }

```

```

(*****
DrawUserItems
*****)

```

```

PROCEDURE DrawUserItems(wPtr: WindowPtr; itemNo: INTEGER);
VAR
  savePen: PenState;
  ik: INTEGER;
  ih: Handle;
  ib: Rect;
BEGIN { DrawUserItems }
  { save, naormalize the pen state }
  GetPenState(savePen);
  PenNormal;

  { frame the list }
  GetDItem(wPtr, iLIST, ik, ih, ib);

```

```

FrameRect(ib);

( frame the default button )
GetDItem(wPtr, iOK, ik, ih, ib);
InsetRect(ib, -4, -4);
PenSize(3, 3);
FrameRoundRect(ib, 16, 16);

( restore the pen's state )
SetPenState(savePen);
END; ( DrawUserItems )

(*****
ReadData: Reads in a number of controls, as specified
by a 'INT#' integer-array resource. Defines new
controls for each.
*****)
FUNCTION ReadData(dPtr: DialogPtr): Boolean;
VAR
  ch:      ControlHandle;
  iah:     ILHandle;
  i:       INTEGER;
  n:       INTEGER;
  ignore:  INTEGER;
  aCell:   Cell;
  p:       Point;
  aRect:   Rect;
BEGIN ( ReadData )
  ( get handle to 'INT#' resource )
  iah := ILHandle(GetResource('INT#', RES_ID));

  IF (iah <> nil) THEN BEGIN
    ( disable list drawing )
    LDraw(false, lh);

    FOR i := 1 TO iah^.count DO BEGIN
      ( create a new control )
      ch := GetNewControl(iah^.int[i], dPtr);

      IF (ch = nil) THEN LEAVE;

      ( make the control visible, without drawing )
      ch^.controlVis := 255;

      ignore := LAddRow(1, 32767, lh);
      SetPt(Point(aCell), 0, i - 1);
      LSetCell(@ch, sizeof(Handle), aCell, lh);
    END; ( for )

    ( fix the controls' controlRects )
    FixCtlRects(lh);

    ( select first control )
    SetPt(Point(aCell), 0, 0);
    LSetSelect(true, aCell, lh);

    ( enable list drawing )
    LDraw(true, lh);
  END; ( iah <> nil )

  ( return TRUE if controls were read OK )
  ReadData := (iah <> nil);
END; ( ReadData )

(*****
InitList: Define and allocate new list, into which
will be placed a number of control handles.
(See ReadData(), above.)
*****)
PROCEDURE InitList(dPtr: DialogPtr);
VAR
  ik:      INTEGER;
  ih:      Handle;
  ib:      Rect;

  dBounds: Rect;
  cSize:   Point;
BEGIN
  ( set up the userItem drawing routine )
  GetDItem(dPtr, iLIST, ik, ih, ib);
  SetDItem(dPtr, iLIST, ik, Handle(@DrawUserItems), ib);

  ( inset rect, for drawing; leave room for scroll bar )
  InsetRect(ib, 1, 1);
  ib.right := ib.right - 15;

  ( allocate space for a single column, no rows )
  SetRect(dBounds, 0, 0, 1, 0);

  ( indicate cell size )
  SetPt(cSize, ib.right-ib.left, 20);

  ( create the list )
  lh := LNew(ib, ( rView
                  dBounds, ( dataBounds )
                  cSize, ( cell size )
                  RES_ID, ( resource ID )
                  dPtr, ( this window )
                  false, ( draw it )
                  false, ( grow box )
                  false, ( sBar, horiz )
                  true); ( sBar, vert )
  END; ( InitList )

  (*****
Cleanup: My name is Cleanup(). You killed my dialog.
Prepare to die!
*****)
PROCEDURE Cleanup(dPtr: DialogPtr);
VAR
  ch:      ControlHandle;
  i:       INTEGER;
  dl:      INTEGER;
  c:       Cell;
BEGIN
  ( default )
  dl := sizeof(Handle);

  ( indicate wait )
  SetCursor(GetCursor(watchCursor));

  ( deallocate the controls from the list )
  FOR i := 0 TO (lh^.dataBounds.bottom - 1) DO BEGIN
    SetPt(Point(c), 0, i);
    LGetCell(@ch, dl, c, lh);
    DisposeControl(ch);
  END; ( for )

  ( dispose of the list )
  LDispose(lh);
  ( dispose of the dialog )
  DisposDialog(dPtr);
END; ( Cleanup )

(*****
Main
*****)
BEGIN
  ( perform the ritual incantation )
  InitGraf(@thePort);
  InitFonts;
  FlushEvents(everyEvent, 0);
  InitWindows;
  InitMenus;
  TEInit;
  InitDialogs(NIL);
  InitCursor;

  ( read in the dialog from its resource template )

```

```

myDialog := GetNewDialog(RES_ID, NIL, POINTER(-1));

( create and init the list of controls )
InitList(myDialog);

IF (ReadData(myDialog)) THEN BEGIN
  ( center and show the dialog )
  CenterWindow(myDialog, true);

  ( save grafPort, make ours current )
  GetPort(savePort);
  SetPort(myDialog);

  ( cycle through ModalDialog() )
  REPEAT
    ModalDialog(@MyFilter, itemHit);
  UNTIL itemHit = iOK;

  ( restore grafPort )
  SetPort(savePort);
END;
( dispose of the dialog's storage )
CleanUp(myDialog);
END. ( FILE Test.p, PROGRAM Test.p )

```

Listing: Test.r

```

/*****
Test.r
Resources for List-Of-Controls example.
*****/
/*****Types*****/
#include "Types.r"
#include "PopupMenuCDEF.r"

type 'INT#' (
  integer = $$Countof(IntegerArray);

  array IntegerArray (
    integer;
  );
);
/*****Defines*****/
#define RES_ID 128
/*****Resources*****/
data 'LDEF' (RES_ID, "ControlLDEF") (
  $$resource("ControlLDEF.LDEF", 'LDEF', 1)
);
data 'CDEF' (pmCDEFResID, "popMenus") (
  $$resource("PopupMenuCDEF.CDEF", 'CDEF', 1)
);
resource 'DLOG' (RES_ID, "RES_ID") (
  {54, 170, 254, 470},
  dBoxProc,
  invisible,
  noGoAway,
  0x0,
  RES_ID,
  ""
);
resource 'DITL' (RES_ID, "RES_ID") (
  ( /* array DITLarray: 3 elements */
    /* [1] */
    {170, 125, 190, 185},
    Button (
      enabled,
      "OK"
    ),
    /* [2] */
    {10, 10, 72, 290},
    UserItem (
      enabled
    ),
    /* [3] */

```

```

    {82, 10, 160, 290},
    StaticText (
      disabled,
      "List of Controls LDEF example. "
      "James Plamondon, Abacus Concepts, "
      "1984 Bonita Avenue, Berkeley, CA "
      "94704."
    )
  );
resource 'INT#' (RES_ID, "Control ID's") (
  ( /* array INTArray: 5 elements */
    /* [1] */
    128,
    /* [2] */
    129,
    /* [3] */
    130,
    /* [4] */
    131,
    /* [5] */
    132
  )
);
resource 'CNTL' (128) (
  {0, 0, 0, 0}, /* rect: controlRect */
  128, /* value: menu rsrc ID */
  invisible, /* vis: standard */
  128, /* max: default menuID */
  2, /* min: default item # */
  popMenuProc/* ProcID: 3 */
  + mCheck, /* var: Check selection */
  0, /* rfCon: for user's use */
  "Thanks to: " /* title: standard */
);
resource 'MENU' (128) (
  128,
  textMenuProc,
  allEnabled,
  enabled,
  "Thanks To: ",
  ( /* 11 items */
    "Mark Williams",
    noIcon, noKey, noMark, plain;
    "Mark Bennet",
    noIcon, noKey, appleChar, plain;
    "Joseph Daniel",
    noIcon, noKey, noMark, plain;
    "Dr. Don Morrison",
    noIcon, noKey, noMark, plain;
    "Andrew Stone",
    noIcon, noKey, noMark, plain;
    "Eleanor Plamondon",
    noIcon, noKey, noMark, plain;
    "Bruce Wampler",
    noIcon, noKey, noMark, plain;
    "Patricia Guffey",
    noIcon, noKey, noMark, plain;
    "Greta Shaw",
    noIcon, noKey, noMark, plain;
    "Monty \"Montana-Unit\" Cole",
    noIcon, noKey, noMark, plain;
    "Dr. Bernard Moret",
    noIcon, noKey, noMark, plain
  )
);
resource 'CNTL' (129) (
  {0, 0, 0, 0}, /* rect: controlRect */
  129, /* value: rsrc ID */
  invisible, /* vis: standard */
  129, /* max: default menuID */
  2, /* min: default item # */
  popMenuProc/* ProcID: 3 */
  + mRes /* var: add res names */

```

```

+ mCheck, /* var: Check selection */
'FONT', /* rfCon: OSType */
'Fonts: " /* title: standard */
);
resource 'MENU' (129) (
129,
textMenuProc,
allEnabled,
enabled,
'Fonts: "
( /* 0 items */
);
resource 'CNTL' (130) (
(0, 0, 0, 0), /* rect: contrlRect */
130, /* value: rsrc ID */
invisible, /* vis: standard */
133, /* max: default menuID */
1, /* min: default item # */
popMenuProc /* ProcID: 3 */
+ mCheck /* var: Check selection */
+ mSub, /* var: sub-menus */
0, /* rfCon: for user's use */
'SubMenu: " /* title: standard */
);
resource 'MENU' (130) (
130,
textMenuProc,
allEnabled,
enabled,
'Root: "
( /* 2 items */
'Root Item1",
noIcon, parent, "\0D131", plain;
'Root Item2",
noIcon, parent, "\0D132", plain
);
resource 'MENU' (131) (
131,
textMenuProc,
allEnabled,
enabled,
"",
( /* 2 items */
'Sub-1 Item1",
noIcon, noKey, noMark, plain;
'Sub-1 Item2",
noIcon, noKey, noMark, plain;
'Sub-1 Item3",
noIcon, parent, "\0D133", plain
);
resource 'MENU' (132) (
132,
textMenuProc,
allEnabled,
enabled,
"",
( /* 2 items */
'Sub-2 Item1",
noIcon, noKey, noMark, plain;
'Sub-2 Item2",
noIcon, noKey, noMark, plain;
'Sub-2 Item3 (a very, very, very wide item)",
noIcon, noKey, noMark, plain
);
resource 'MENU' (133) (
133,
textMenuProc,
allEnabled,
enabled,
"
```

```

( /* 2 items */
'Sub-3 Item1",
noIcon, noKey, noMark, plain;
'Sub-3 Item2",
noIcon, noKey, noMark, plain;
'Sub-3 Item3",
noIcon, noKey, noMark, plain
);
resource 'CNTL' (131) (
(0, 0, 0, 0), /* rect: contrlRect */
134, /* value: menu rsrc ID */
invisible, /* vis: standard */
134, /* max: default menuID */
1, /* min: default item # */
popMenuProc /* ProcID: 3 */
+ mCheck, /* var: Check selection */
0, /* rfCon: for user's use */
'At Abacus: " /* title: standard */
);
resource 'MENU' (134) (
134,
textMenuProc,
allEnabled,
enabled,
'At Abacus: "
( /* 5 items */
'Dan Feldman",
noIcon, noKey, noMark, plain;
'Jim Gagnon",
noIcon, noKey, noMark, plain;
'Will Scoggin",
noIcon, noKey, noMark, plain;
'James Plamondon",
noIcon, noKey, noMark, plain;
'Jay Roth",
noIcon, noKey, noMark, plain;
'Tiffany Fyans",
noIcon, noKey, noMark, plain;
'Jeanette Stafford",
noIcon, noKey, noMark, plain
);
resource 'CNTL' (132) (
(0, 0, 0, 0), /* rect: contrlRect */
135, /* value: menu rsrc ID */
invisible, /* vis: standard */
135, /* max: default menuID */
1, /* min: default item # */
popMenuProc /* ProcID: 3 */
+ mCheck, /* var: Check selection */
0, /* rfCon: for user's use */
'Programs: " /* title: standard */
);
resource 'MENU' (135) (
135,
textMenuProc,
allEnabled,
enabled,
'Programs: "
( /* 5 items */
'StatView",
noIcon, noKey, noMark, plain;
'StatView 512+",
noIcon, noKey, noMark, plain;
'StatView SE+Graphics",
noIcon, noKey, noMark, plain;
'StatView II",
noIcon, noKey, noMark, plain;
'SuperANOVA",
noIcon, noKey, noMark, plain
);
);
```



Pascal Procedures

Build a Network Painting Program



*Edgar Circenis
Rod Magnuson
Lincoln, NE
Volume 5, Number 7*

[Edgar Circenis: Has programmed the Mac since 1984. Has been employed at Nordic Software and written software for both the Apple II and Macintosh. Graduate of the University of Nebraska—Lincoln with a B.S. C.S. Rod Magnuson: Has had 2 years programming the Macintosh. Founded MindVision software with Steve Kiene in early 1988. Co-author of MaxWrite, the first color word processor and MindVision's first offering.]

JamPaint: A Networked Paint Program **JamPaint's Beginnings**

Back in 1987, an AppleTalk network was installed in our workplace. We decided to write a simple program to allow communications between different nodes of this network. Our program, Transceive, used the NBP and LAP protocols to send text messages between nodes.

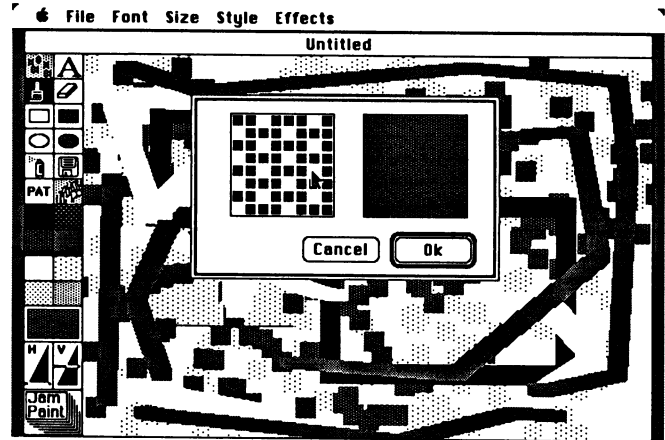
When we got the program working and realized how easy it was to use AppleTalk, we took Transceive one step further by rewriting it so that a user on one Mac could draw on the screen of several other Macs. To do this, we created a message format consisting of a message type identifier and data needed for the given type. We implemented two different types of messages: Click and Drag. A click message consisted of the Click message identifier and a Point. The drag message took a similar form. A Click message was sent when a user initially clicked on the screen. Then, Drag messages were sent every time that the mouse location changed, until the mouse button was released.

Transceive was rewritten to operate in either master or slave mode. In master mode, the click and drag messages were generated and broadcast over the network. In slave mode, the messages were interpreted and the results were drawn on the screen.

We relied entirely on the LAP protocol for our transmissions. This caused a problem because LAP has no facility for buffering incoming messages. Therefore, we experimented with using a VBLTask to periodically check the message buffer and put messages into our own buffer. This was a bad idea because the LAP protocol moves memory and VBLTasks cannot cause memory moves. Our program crashed periodically because of this. To solve our problem, we had to resort to polling the message buffer. This caused us to lose messages on occasion, but it was better than periodic crashes.

Finally, we were able to have up to eight (a practical limit) Macs hooked up and drawing on each other's screens simultaneously. This was great fun for the staff at work, so we thought about the feasibility of extending Transceive into more of a paint program.

Transceive sat idle for months while we thought about what could and couldn't be done easily over a network. We wanted to keep message sizes small and had to have a system where lost



messages wouldn't cause problems or crashes (with polling, messages will get lost). We thought about using the ATP protocol to avoid message loss, but decided that response time would be too slow for a real-time paint program. Eventually, we decided that LAP was still the best choice. Then, we decided to write JamPaint.

JamPaint is a simple, first come, first served Networked Paint program. To control users, we have an array of environment records. Each slot in this array is filled as unidentified users announce themselves (by sending an initial message) on the network. Again, we decided to limit the number of users to eight to reduce traffic on the network. JamPaint was written during a long weekend, but fixing bugs and adding new tools has kept us busy for 3-4 months. Initially, we had decided to make JamPaint a color paint program for the Mac II in addition to being networked. We wrote color code, but never spent much time trying to get offscreen pixmaps to work correctly. Hence, our screen updating was a mess on a Mac II. As soon as we get that code working, we will re-release JamPaint for color use on the Mac II.

The Way It was Programmed

JamPaint was written in MPW Pascal with little regard for proper programming technique. As a result, JamPaint variables are for the most part global variables. By using many globals, we eliminated the creation and initialization of large stack frames. Most of our procedure parameters are VAR parameters (to eliminate excessive stack frame initialization). By adopting this style, we were able to decrease memory usage and increase execution speed of our program.

JamPaint implements eight paint tools: paint brush, letters tool, eraser, rectangle, filled rectangle, oval, filled oval, and a spray-can tool. Our palette includes eight patterns which can be edited by double-clicking on them. JamPaint also includes a

unique pen sizing tool which allows horizontal and vertical pen size to be set independently. We implemented the Pattern and Constraint keys found in most paint programs and tried to emulate the standard paint program user interface.

Once again, we used the type-and-data format for sending our messages across AppleTalk. To make JamPaint a fully networked paint program, we needed to create many new message types. For instance, occurrences such as selecting a new tool had to be transmitted over the network so that each node would know how to handle subsequent drag messages. A problem with this approach is that a tool selection message may be lost and drag message may be mis-interpreted. Fortunately, with only eight nodes, this doesn't happen often.

With JamPaint, we eliminated the master/slave modes used in Transceive and put all nodes in a combined master/slave mode. What this means is that every message incoming from a previously identified node will be executed. Also, every action performed locally will be broadcast.

To handle the transmission and execution of command messages, we implemented a simple command queue. Every incoming message is put into this queue, and periodically, the first message in the queue is executed. To save ourselves the trouble of duplicating this mechanism for local actions, and to simplify the testing of the incoming message routines, outgoing messages are broadcast and then put into the incoming message queue for execution locally. This way, all drawing gets handled by the same set of routines. Our approach not only cut down on code size, but simplified debugging and streamlined our message handling system.

How Things Get Drawn

JamPaint uses an offscreen bitmap to perform its updating and redraw. All drawing commands are duplicated — once for the screen, and once for an offscreen bitmap. When an update is required, CopyBits is used to transfer the image back onto the screen. This approach was made necessary when we decided that our window wouldn't always be the front window (because of DA's, dialogs, etc.).

The Users on the Network

JamPaint treats each user as a paint environment. When a user first starts using JamPaint, a message corresponding to their first action is broadcast over the network. Any nodes that still have room in their environment tables (remember, only eight entries) will notice that this message comes from an unidentified node. They will then request that the sending node transmit its environment. The same happens in the other direction. The new node will not recognize the already operating nodes and will fill its own environment tables the same way.

The following data structure is used to keep track of a user environment:

```
UserTableRec = record
  id:integer;           ( AppleTalk ID number )
  time:longint;         ( time elapsed since last message sent )
  theTool:integer;      ( current pain tool )
  x,y:integer;          ( last pen coordinate )
  theMode:boolean;      ( line or dot mode )
  thePat:pattern;       ( penPat )
  theClr:RGBColor;      ( color for Mac II )
  hSize,vSize:integer; ( pen size )
```

```
theFnum:integer;       ( font number )
theFsize:integer;      ( font size )
theFstyle:style;       ( font style )
splatRad:integer;      ( splatter radius for airbrush )
splatSpeed:integer;    ( splatter speed for airbrush )
```

end;

id : The AppleTalk node id of the user. This is needed to send our messages through the LAP protocol calls.

time : The TickCount value of the last time a message from the user was executed. This is used to eliminate "dead" users from the network.

theTool : is the current tool for the user.

x, y : The current pen position of the user.

theMode : The current mode of the user. JamPaint supports two drawing modes: line and dot. In line mode, drag packets connect the current x,y position with a line to the previous position. In Dot mode, no connection is made.

splatRad : The airbrush radius; double clicking on the spray-can tool will bring up a dialog that allows the airbrush radius to be resized.

splatSpeed : The airbrush speed, or how many particles get sprayed each time a drag message is received; this can also be changed by double-clicking the spray-can tool.

Packet Types

The following is the variant data structure used for the different message types in JamPaint:

```
MSGType=(paintMode,alpha,settool,eraseall,setpat,setcolor,setpen,
setfont, setpos,drag,rectpck, setSplat,rqinfo,sendinfo);
```

```
LAPMsg = record
  size :packed array [0..1] of byte;
  theType :MSGType;
  id :byte;
  case MSGType of
    paintMode:(mode:boolean);
    alpha:(ch:char); ( type char in current font,size,style ,location )
    settool:(tool:integer); ( select a tool )
    setpat:(pat:pattern); ( select a pattern )
    setcolor:(clr:RGBColor); ( select a color )
    setpen:(px,py:integer); ( set pen size to px,py )
    setfont:(fnum:integer; ( select font characteristics )
      fsize:integer;
      fstyl:style);
    setpos:(mx,my:integer); ( set current position to mx,my )
    drag:(cx,cy:integer); ( mouse dragged from mx,my to cx,cy )
    setSplat:(sRad:integer; ( resize splatter tool )
      sSpeed:integer);
    rectpck:(rct:rect; ( rectangle for TRect..TF0val )
      optDown:boolean);
    sendinfo:(info:UserTableRec); ( user info )
  end;
```

All paint functions are executed by transmitting one of these message data structures to the users on the network.

Implementing Tools That Require Waiting

Implementing the oval and rectangle drawing tools was more difficult than implementing the other tools. This is due to

the fact that these objects can be enlarged and shrunk before actually being placed into a document. With messages coming in from other machines, we had to address the problem of what to do while the user was sizing an object. Also, we had to worry about how resizing was to be done on a remote machine.

We decided that it would be a waste of code and time to resize objects on a remote screen. After all, the finished object was what we were after. This left only the problem of how to handle resizing locally while executing incoming messages. Our solution was to do all resizing on the screen only. We would leave the offscreen bitmap alone until the object was complete. All incoming messages get executed on the offscreen bitmap and updated to the screen. Then, when the sized object is complete, a message is transmitted to draw it both locally and elsewhere. This way, we reduce AppleTalk traffic, eliminate the problem of missed messages, and avoid over-writing of previous drawing.

Drag Packets

Whenever a user is holding down the mouse, drag messages are transmitted (unless the rectangle or oval tools are selected). Drag messages contain only the new location of the mouse. When a Drag message is received, it is sent to one of the drawing routines depending on the current tool in the environment record for the user in question. This local interpretation of commands was chosen because it results in smaller AppleTalk packets, leading to better throughput.

JamPaint's Future

Is there a future for JamPaint? We think so. Anyone who has played (yes, it's more like playing than painting) with JamPaint has to admit that it was fun. We plan to enhance JamPaint by making it into a real paint program. Planned enhancements include:

- Color
- Separate palette and tool windows with more patterns and tools.
- A full page (or possibly oversize) drawing surface.
- Selection tools.
- Special effects (fades, contrast enhancement, smoothing, etc.).
- Paint channels and privacy mode.
- Object-oriented drawing.
- Enhance compatibility with existing paint programs.
- Memos.

We would like to think that JamPaint is a step into the future. JamPaint is a new kind of tool which not only allows many people to see the same piece of work, but to also modify it. We are trying to bring real connectivity to the Mac by creating multi-user software.

We ask that you do not change JamPaint and distribute it. It has been copy righted. To maintain some sort of order, and minimize confusion, MindVision Software will gladly accept suggestions for improvements to this program. If you would like to add something to JamPaint, let us know what you're doing and maybe we can work together. Our About Window has room for

plenty of names; just don't expect to become rich.

After Thought

We suggest that you run JamPaint on System 4.1 or 4.2 because JamPaint is very slow running under System Software 6.0. We do not know why this is. We haven't had time to look into it yet.

We have also noted that Apple has implemented something similar to JamPaint in HyperCard with their XCMD set that allows access to NBP and LAP AppleTalk protocols from script.

Listing: MakeFile.p

```
***** JamPaint MakeFile *****
Pascal = Pascal
POptions = -s
```

```
.p.o      f .p
Pascal -k "(Libraries)" (default).p
```

```
JamPaint      f JamPaint.r JamPaint.code
Rez "(RIncludes)"Types.r JamPaint.r -o JamPaint -c JPNT
SetFile -a B -d . JamPaint
JamPaint.code f JamPaint.p.o
Link JamPaint.p.o @
"(Libraries)"Interface.o @
"(Libraries)"Paslib.o @
"(Libraries)"Runtime.o @
-o JamPaint.code
```

Listing: JamPaint.p

Program JamPaint;

```
(*****
JamPaint - the Network Paint Application
*****
Copyright 1988 by Edgar Circenis and Rod Magnuson
All Rights Reserved
*****
Started: 12/26/87 Revision: 5/21/88
*****
```

Problem areas:

- Do we need to worry about nets as well as nodes?
- How big should PQ be?
- VBLTasks cannot move memory.
- Change xEnqueue and xDequeue to access PQ directly.
- What causes initial LAPWrite error 070 (err = -95)???
- NOTE: if we end up using polling, a queue will be unnecessary.
- use VAR parameters where it will speed things up.

USES

```
($LOAD MacDump)
MemTypes, Quickdraw, OSIntf, ToolIntf, PackIntf, PickerIntf,
Script,
($LOAD AppleTalkDump)
AppleTalk,
($LOAD MacPrintDump)
PrintTraps;
```

CONST

```
lastMenu      = 7;
maxUsers      = 8;      ( User Table size )
OurType       = 39;      ( a random LAP protocol type )
listSize      = 550;     ( incoming packet queue size )
VBLcnt        = 2;      ( myTask.vblCount )
connect       = false;
spray         = true;
idleTime      = 60*60*5; ( five minute idle time )
eraseCursor   = 128;
splatCursor   = 8088;    ( splatter cursor ID )
( — TOOLS — )
```

```

Tnone = 0;      ( no tool, or special tool )
Tspray = 1;     ( not really a tool! )
Tletters = 2;   ( letter tool )
Tbrush = 3;     ( paint brush )
Terase = 4;     ( eraser )
Trect = 5;      ( Rectangle )
Tfrect = 6;     ( Filled Rectangle )
TOval = 7;      ( Oval )
TFoval = 8;     ( Filled Oval )
TSplatter = 9;  ( Splatter Tool )
TDisk = 10;     ( Disk Access )
Tpat = 11;      ( not a tool )
Tcolor = 12;    ( not a tool )
( — DEFAULTS — )
dSize = 12;
dStyle = [1];
dPen = 1; ( 1 x 1 pen )
dColor = 1; ( black )
dTool = Tbrush;
dMode = connect;
dsplatterCount = 30; ( * default splatter speed * )
dsplatRad = 20; ( * default splatter radius * )
( **** make a decision about this and delete it **** )
polling = true;

appleMenu = 1;
fileMenu = 2;
fontMenu = 3;
sizeMenu = 4;
styleMenu = 5;
effectsMenu = 6;
editMenu = 7;

```

TYPE

```

UserTableRec = record
    id      :integer;    ( appleTalk id )
    time    :longint;    ( idle time )
    theTool :integer;    ( current tool )
    x,y     :integer;    ( mouse coordinates )
    theMode :boolean;    ( paint mode )
    thePat  :pattern;    ( current pattern )
    theClr  :RGBColor;   ( current color )
    hSize,vSize:integer; ( penSize )
    theFnum :integer;    ( font number )
    theFsize:integer;    ( font size )
    theFstyle:style;     ( font style )
    splatRad :integer;   ( splatter's radius )
    splatSpeed:integer;  ( splatter's speed )
end;

MSGType = (paintMode,alpha,settool,eraseall,setpat
,setcolor,setpen,setfont,setpos,drag,rectpck,setStatus,sendinfo);
LAPMsg = record
    size :packed array [0..1] of byte; (packet length)
    theType :MSGType;    ( message type )
    id :byte;            ( node id )
case MSGType of
    paintMode: (mode :boolean);
    ( select paint mode type: connect, spray )
    alpha: (ch :char);
    ( type char in current font, size, style, location )
    settool: (tool :integer); ( select a tool )
    setpat: (pat :pattern); ( select a pattern )
    setcolor: (clr:RGBColor); ( select a color )
    setpen: (px,py:integer); (set pen size to px,py )
    setfont: (fnum :integer; ( font characteristics )
        fsize:integer;
        fstyle:style);
    setpos: (mx,my:integer); (set position to mx,my)
    drag: (cx,cy:integer);
    ( mouse dragged from mx,my to cx,cy )
    setStatus: (sRad:integer; ( resize splatter tool )
        sSpeed:integer);
    rectpck: (rct:rect; (rectangle for TRect..TFoval)

```

```

        optDown:boolean);
        sendinfo: (info :UserTableRec); ( user info )
    end;
LAPMsgPtr = ^LAPMsg;
PacketList = Packed array [0..listSize-1] of LAPMsg;
PacketQueue = record
    head :integer;
    tail :integer;
    queue :PacketList;
end;
PacketQueuePtr = ^PacketQueue;

CArray = Array [1..8] of RGBColor;
CPtr = ^CArray;
CHandle = ^CHandle;
BitMapPtr = ^BitMap;

```

VAR

(A NOTE ABOUT GLOBALS AND SUCH:

Everyone is taught that using global variables is a sin. We do not agree. By using many globals and declaring parameters in parameter lists to be VAR parameters (even when not needed), we increase the speed of our code by reducing stack frame size and the creation of local copies of variables.)

```

myEvent      :EventRecord;
theItem,theMenu,refnum:integer;
theChar      :Char;
code         :integer;
tempWindow,myWindow :WindowPtr;
doneflag     :boolean;
PrDebug      :boolean;
mods         :longint;
mymenus      :array [1..lastmenu] of MenuHandle;
FileMenuPresent :boolean;
err          :integer;
dlg          :DialogPtr;
itype        :integer;
item         :handle;
box          :rect;
itemHit      :integer;
myTask       :VBLTask;
UT           :array [0..maxUsers] of UserTableRec;
PQ           :PacketQueue;
myNode,myNet :integer;
LAPPrh,LAPWh :ABRCHandle;
LAPPrbuf,LAPWbuf :LAPMsg;
DrawWindow   :WindowPtr;
direct,prect :rect;
jamPic,palette :picHandle;
ToolRects    :array [1..20] of Rect;
curPatRect   :rect;
hSizeRect,vSizeRect :rect;
PatternsUp   :Boolean;
thePatterns  :array [1..8] of pattern;
theColors    :CArray;
curPat,curColor :integer;
theFontidx   :integer;
theSizeidx   :integer;
clickTime    :longint;
lastTool     :integer;
theECurs,theCurs :cursor;
arrowCurs,updateCurs :boolean;
bmap,Wbits   :bitMap;
oldRgn,oldClip :RgnHandle;
changed,saved :boolean;
fVref        :integer;
MultiFinderRunning :boolean;
ColorQDrawImplm :boolean;
MacII,PixDraw :boolean;
myCGrafPtr   :CGrafPtr;
myCGrafPort  :CGrafPort;
ourCMHandle  :CTabHandle;
offpix,Wpix  :PixMapHandle;
theHand,theSizer :CursHandle;
theSprayer,thePlacer :CursHandle;

```

```

hPrint          :THPrint;

Procedure SendLAP(who:byte); Forward;
Procedure CheckRead; Forward;
Procedure DrawContents; Forward;

Procedure Debug(s:str255);
begin
  if PrDebug then
    begin
      PrCtlCall(iPrIOCtl,ord(0s)+1,length(s),0);
      PrCtlCall(iPrDevCtl,$0003FFFF,0,0);
    end;
end;

Function Str(i:longInt):str255;
( These two functions are invaluable )
begin
  NumToString(i,Str);
end;

Function Val(s:str255):longInt;
begin
  StringToNum(s,Val);
end;

function MyGetNextEvent(evtMask:Integer;VAR
Evt:EventRecord):Boolean;
( * This allows us to be more MultiFinder compatible * )
begin
  if MultiFinderRunning then
    MyGetNextEvent:=WaitNextEvent(evtMask,Evt,15,Nil)
  else
    begin
      SystemTask;
      MyGetNextEvent:=GetNextEvent(evtMask,Evt);
    end;
end;

Procedure MenuString(s:str255); (debugging simple stuff )
var
  dMenu :MenuHandle;
begin
  s := Concat(s,' <click>');
  DMenu := NewMenu(999,s);
  InsertMenu(DMenu,0);
  DrawMenuBar;
  sysBeep(1);
  repeat Until
MyGetNextEvent(mdownMask+keyDownMask+AutoKeyMask,myEvent);
  DeleteMenu(999);
  DisposeMenu(DMenu);
  DrawMenuBar;
end;

Function idleFilter(item:Integer;theDlg:DialogPtr):Integer;
begin
  SetUpA5;
  idleFilter:=item;
  CheckRead;
  RestoreA5;
end;

procedure HiliteButton(theDialog:DialogPtr);
begin
  SetPort(theDialog);
  GetDItem(theDialog,ok,iType,iTem,Box);
  InsetRect(Box,-4,-4);
  PenSize(3,3);
  FrameRoundRect(box,15,16);
  PenNormal;
end;

Procedure DoErr(i:integer);

```

```

begin
  if err<>0 then
    begin
      if prDebug then
        Debug(concat('*** ',str(i),': error = ',str(err)))
      else
        menuString(concat(str(i),': error = ',str(err)));
      end;
    end;

Procedure RsrcErr;
begin
  if ResError<>0 then
    begin
      if PrDebug then
        Debug(concat('*** RsrcErr=',str(ResError)))
      else
        menuString(concat('RsrcErr=',str(ResError)));
      end;
    end;

Procedure SetUpMenus;
var
  i :integer;
  s,fName :str255;
begin
  InitMenus;
  FileMenuPresent:=false;
  for i := 1 to lastmenu do
    MyMenus[i] := GetMenu(254+i);
  for i := 1 to lastmenu-1 do
    InsertMenu(myMenus[i],0);
  AddResMenu(MyMenus[appleMenu], 'DRVr');
  AddResMenu(MyMenus[fontMenu], 'FONT');
  GetFontName(app1Font,fName);
  for i := 1 to countMItems(mymenus[fontMenu]) do
    begin
      GetItem(mymenus[fontMenu],i,s);
      if s=fName then
        begin
          CheckItem(mymenus[fontMenu],i,true);
          theFontidx := i;
          leave;
        end;
      end;
    end;
  for i:=1 to 9 do
    begin
      GetItem(myMenus[sizeMenu],i,s);
      if RealFont(theFontidx,Val(s))
      then
        SetItemStyle(myMenus[sizeMenu],i,[Outline])
      else
        SetItemStyle(myMenus[sizeMenu],i,[1]);
      end;
    end;
  DrawMenuBar;
end;

Procedure xEnqueue(var msg:LAPMsg); ( add queue element )
begin
  Debug('xEnqueue');
  with PQ do
    if (tail+1) mod listSize = head then
      begin
        sysbeep(1); ( oops, queue full! )
        Debug('Queue Full');
      end
    else
      begin
        queue[tail] := msg;
        tail := (tail+1) mod listSize;
        Debug(concat(' head=',str(head),', tail=',str(tail)));
      end;
    end;
end;

```

```

Function xDequeue(var msg:LAPMsg):boolean; ( remove first
queue element )
begin
  with PQ do
    if tail=head then
      xDequeue := false
    else
      begin
        Debug('xDequeue: dequeued');
        msg := queue[head];
        head := (head+1) mod listsize;
        xDequeue := true;
        Debug(concat(' head=',str(head),', tail=',str(tail)));
      end;
    end;
end;

procedure DoPageSetUp;
var
  tempPrint : THPrint;
  err       : OSErr;
begin
  PrOpen;
  If PrError=noErr then
    begin
      tempPrint:=hPrint;
      err:=HandToHand(handle(tempPrint));
      Repeat
        if PrStlDialog(tempPrint) then
          begin
            DisposHandle(handle(hPrint));
            If MemError<>NoErr then
              SysBeep(1);
            hPrint:=tempPrint;
            err:=HandToHand(handle(hPrint));
            if err<>NoErr then
              SysBeep(1);
          end;
        Until not PrValidate(hPrint);
      end;
      PrClose;
      DisposHandle(handle(tempPrint));
    end;
end;

Function FindUser(id:integer):integer; ( find UT entry )
( find user in UT. If not in UT, return -1 and send
<rqinfo> packet to user if table not yet full. If data is not
valid, returns -(index+1). )
var
  i :integer;
begin
  Debug('FindUser');
  for i := 0 to maxUsers do ( CASE 1: look for user )
    if UT[i].id=id then
      begin
        FindUser := i; ( valid user found )
        Debug(' user found');
        exit(FindUser);
      end;
  Debug(' user not found');
  for i := 1 to maxUsers do (CASE 2:look for empty slot)
    if UT[i].id=0 then
      begin
        UT[i].id := id;
        LAPwbuf.theType := rqinfo;
        SendLAP(id);
        FindUser := -1; ( user not valid (yet) )
        exit(FindUser);
      end;
  for i := 1 to maxUsers do (CASE 3:look for idle user)
    if tickCount-UT[i].time>idleTime then
      begin
        UT[i].id := id;
        LAPwbuf.theType := rqinfo;
        SendLAP(id);

```

```

        FindUser := -1; ( user not valid (yet) )
        exit(FindUser);
      end;
end;

end;

Procedure SetUserState(j:integer); ( set previous state )
begin
  Debug('SetUserState');
  with UT[j] do
    begin
      MoveTo(x,y);
      PenPat(thePat);
      if MacII then
        RGBForeColor(theClr);
      PenSize(hSize,vSize);
      TextFont(theFnum);
      TextSize(theFsize);
      TextFace(theFstyle);
    end;
  end;
end;

Procedure OffBits; ( set offscreen bitmap )
begin
  if MacII then
    begin
      if PixDraw then
        begin
          SetPort(GrafPtr(myCGrafPtr));
        end;
    end
  else
    begin
      oldRgn:=NewRgn;
      CopyRgn(DrawWindow^.visRgn,oldRgn);
      RectRgn(DrawWindow^.visRgn,bmap.bounds);
      Wbits:=DrawWindow^.portbits;
      SetPortBits(bmap);
    end;
  end;
end;

Procedure OnBits; ( set onscreen bitmap )
begin
  if MacII
  then
    begin
      if PixDraw then
        begin
          SetPort(DrawWindow);
        end;
    end
  else
    begin
      CopyRgn(oldRgn,DrawWindow^.visRgn);
      DisposeRgn(oldRgn);
      SetPort(DrawWindow);
      SetPortBits(Wbits);
    end;
  end;
end;

end;

Procedure DoSplatter(user:integer);
var
  r :rect;
  i,xx,yy :integer;
begin
  ClipRect(direct);
  with UT[user] do
    for i := 1 to splatSpeed do
      begin
        repeat
          xx := random mod SplatRad;
          yy := random mod SplatRad;
        until xx*xx+yy*yy <= SplatRad*SplatRad;
        xx := xx + x;
        yy := yy + y;

```

```

        SetRect(r,xx,yy,xx+hSize,yy+vSize);
        PaintRect(r);
        OffBits;
        PaintRect(r);
        OnBits;
    end;
    ClipRect(drawWindow^.portRect);
end;

Procedure DoRect(utnum:integer;r:rect;optDown:boolean);
( handle rect packets )
begin
    Debug('DoRect');
    with UT[utnum] do
        case theTool of
            TRect: begin
                if not optDown then
                    PenPat(black);
                    FrameRect(r);
                    OffBits;
                    FrameRect(r);
                    OnBits;
                end;
            TFRect: begin
                PaintRect(r);
                OffBits;
                PaintRect(r);
                OnBits;
                if not optDown then
                    begin
                        PenPat(black);
                        FrameRect(r);
                        OffBits;
                        FrameRect(r);
                        OnBits;
                    end;
                end;
            T0val: begin
                if not optDown then
                    PenPat(black);
                    FrameOval(r);
                    OffBits;
                    FrameOval(r);
                    OnBits;
                end;
            TF0val: begin
                PaintOval(r);
                OffBits;
                PaintOval(r);
                OnBits;
                if not optDown then
                    begin
                        PenPat(black);
                        FrameOval(r);
                        OffBits;
                        FrameOval(r);
                        OnBits;
                    end;
                end;
            end;
        end;
    end;

Procedure DoDrag(utnum,dx,dy:integer); (drag packets)
var
    pt :point;
    r :rect;
begin
    Debug('DoDrag');
    with UT[utnum] do
        case theTool of
            Tbrush: begin
                if theMode=connect then
                    begin
                        GetPen(pt);

```

```

                        LineTo(dx,dy);
                        OffBits;
                        MoveTo(pt.h,pt.v);
                        LineTo(dx,dy);
                        OnBits;
                    end
                else
                    begin
                        MoveTo(dx,dy);
                        Line(0,0);
                        OffBits;
                        MoveTo(dx,dy);
                        Line(0,0);
                        OnBits;
                    end;
                x := dx;
                y := dy;
            end;
            Terase: begin
                penPat(white);
                if theMode=connect then
                    begin
                        GetPen(pt);
                        LineTo(dx,dy);
                        OffBits;
                        MoveTo(pt.h,pt.v);
                        LineTo(dx,dy);
                        OnBits;
                    end
                else
                    begin
                        MoveTo(dx,dy);
                        Line(0,0);
                        OffBits;
                        MoveTo(dx,dy);
                        Line(0,0);
                        OnBits;
                    end;
                x := dx;
                y := dy;
            end;
            TSplatter: begin
                x := dx;
                y := dy;
                DoSplatter(utnum);
            end;
        end;
    end;

Procedure SendMyInfo(id:integer); ( send UT to node <id> )
begin
    Debug('SendMyInfo');
    with LAPwbuf do
        begin
            theType := sendinfo;
            info := UT[0];
        end;
        SendLAP(id);
    end;

Procedure DebugLAPType(t:MSGType);
begin
    case t of
        paintMode: Debug(' paintMode');
        alpha: Debug(' alpha');
        settool: Debug(' settool');
        eraseall: Debug(' eraseall');
        setpat: Debug(' setpat');
        setcolor: Debug(' setcolor');
        setpen: Debug(' setpen');
        setfont: Debug(' setfont');
        setpos: Debug(' setpos');
        drag: Debug(' drag');
        rectpck: Debug(' rect');

```

```

    rqinfo: Debug('rqinfo');
    sendinfo: Debug('sendinfo');
end;
end;

Procedure ExecMessage(var msg:LAPMsg); ( handle a LAP
message )
var
    j      :integer;
    pt     :point;
    savePort :GrafPtr;
begin
    Debug('ExecMessage');
    changed := true;
    with msg do
        if theType=rqInfo then ( handle rqInfo requests )
            begin
                SendMyInfo(id);
                DebugLAPType(theType);
            end
        else ( handle graphics commands )
            begin
                GetPort(savePort);
                SetPort(DrawWindow);
                j := FindUser(id); ( get index into user table )
                if j=>0 then
                    begin
                        SetUserState(j); ( set user's port state )
                        with UT[j] do
                            begin
                                DebugLAPType(theType);
                                case theType of
                                    paintMode: theMode := mode;
                                    alpha: begin
                                        GetPen(pt);
                                        DrawChar(ch);
                                        OffBits;
                                        MoveTo(pt.h,pt.v);
                                        DrawChar(ch);
                                        OnBits;
                                        GetPen(pt);
                                        x := pt.h;
                                        y := pt.v;
                                    end;
                                    eraseall: begin
                                        eraseRect(direct);
                                        OffBits;
                                        eraseRect(direct);
                                        OnBits;
                                    end;
                                    settool: theTool := tool;
                                    setpat: thePat := pat;
                                    setcolor: theClr := clr;
                                    setpen: begin
                                        hSize := px;
                                        vSize := py;
                                    end;
                                    setfont: begin
                                        theFnum := fnum;
                                        theFsize := fsize;
                                        theFstyle := fstyle;
                                    end;
                                    setpos: begin
                                        x := mx;
                                        y := my;
                                    end;
                                    drag: DoDrag(j,cx,cy);
                                    rectpck: DoRect(j,rct,optDown);
                                    setSplat: begin
                                        splatRad := sRad;
                                        splatSpeed := sSpeed;
                                    end;
                                    sendinfo: begin
                                        UT[j] := info;

```

```

                                Debug(' sendInfo received');
                                end;
                                time := tickCount; ( user is active )
                                end;
                            else
                                Debug(' msg not executed');
                                SetPort(savePort);
                            end;
                        end;
                    end;
                end;
            end;
        end;
    end;

Procedure CheckQueue; (execute packet from incoming queue)
var
    msg :LAPMsg;
begin
    if polling then
        CheckRead;
    if xDequeue(msg) then ( incoming queue )
        ExecMessage(msg);
    end;

Procedure SendLAP(who:byte); (send LAP Packt to node <id>)
var
    sz :longint;
begin
    Debug('SendLAP');
    DebugLAPType(LAPwbuf.theType);
    (XXXXXXXXXX THIS NEEDS TO BE RECALCULATED! XXXXXXXXXXXX)
    case LAPwbuf.theType of
        paintMode,alpha,settool: sz := 2;
        eraseall: sz := 0;
        setpat: sz := sizeof(pattern);
        setcolor: sz := sizeof(RGBColor);
        setfont: sz := sizeof(style) + 4;
        setpos,setpen,drag: sz := 4;
        rectpck: sz := sizeof(rect)+2;
        sendinfo: sz := sizeof(UserTableRec);
        setSplat: sz := 4;
    end;
    sz := sizeof(MSGType)+4+sz; ( LAP packet size )
    sz := sizeof(LAPMsg); ( <- DEBUGGING ONLY )
    with LAPwbuf do ( set up LAP packet size, sender id )
        begin
            id := myNode;
            size[0] := sz div 256;
            size[1] := sz mod 256;
        end;
        with LAPwh^^ do
            begin
                lapAddress.lapProtType := OurType; (protocol type)
                lapAddress.dstNodeID := who; ( destination node )
                lapReqCount := sz; ( packet size )
                lapDataPtr := @LAPwbuf; ( packet data pointer )
            end;
            err := LAPWrite(LAPwh,false); ( send LAP packet )
            DoErr(70);
            if LAPwbuf.theType < rqInfo then
                ( don't want to execute rqInfo or sendInfo locally )
                ExecMessage(LAPwbuf); ( execute message locally )
            end;
        end;

Procedure CheckRead;
var
    destnode :integer;
begin
    SetUpA5;
    err := LAPRh^^.abResult;
    if err=>0 then
        begin
            Debug('CheckRead: msg rcvd');
            DebugLAPType(LAPrbuf.theType);
            destNode := LAPRh^^.lapAddress.dstNodeID;
            if (destNode=255) or (destNode=myNode) then

```

```

        xEnqueue(LAPrbuf);
    end;
    if err < 1 then
    begin
        with LAPrh^^ do
        begin
            lapAddress.lapProtType := OurType;
            lapReqCount := Sizeof(LAPMsg);
            lapDataPtr := @LAPrbuf;
        end;
        err := LAPRead(LAPrh, true);
        DoErr(5000);
    end;
    myTask.vblCount := VBLCnt;
    RestoreA5;
end;

Procedure SetUpRead;
begin
    with LAPrh^^ do
    begin
        lapAddress.lapProtType := OurType;
        lapReqCount := Sizeof(LAPMsg);
        lapDataPtr := @LAPrbuf;
    end;
    err := LAPRead(LAPrh, true);
    DoErr(71);
    if not polling then
    begin
        with myTask do
        begin
            qType := ord(vType);
            vblAddr := @CheckRead;
            vblCount := VBLCnt;
            vblPhase := VBLCnt div 2 + 1;
        end;
        err := VInstall(@myTask);
        DoErr(500);
    end;
end;

Procedure DisplayPatterns;
var
    i : integer;
begin
    for i := 1 to 8 do
        FillRect(toolRects[i+12], thePatterns[i]);
        FillRect(curPatRect, thePatterns[curPat]);
    end;
end;

Procedure DisplayColors;
var
    i : integer;
begin
    PenNormal;
    for i := 1 to 8 do
    begin
        RGBForeColor(theColors[i]);
        PaintRect(toolRects[i+12]);
    end;
    RGBForeColor(theColors[curColor]);
    PaintRect(curPatRect);
end;

Procedure DrawPalette;
var
    r : rect;
begin
    SetPort(DrawWindow);
    DrawPicture(palette, prect);
    with UT[0] do
    begin
        InvertRect(toolRects[theTool]);
        if not theMode then

```

```

        with toolRects[theTool] do
        begin
            PenMode(PatXOR);
            PenPat(1tgray);
            PaintRect(toolRects[Tspray]);
            PenNormal;
        end;
    end;
    PenMode(PatXOR);
    PenPat(1tgray);
    if PatternsUp then
    begin
        PaintRect(toolRects[Tcolor]);
        PenNormal;
        DisplayPatterns;
    end
    else
    begin
        PaintRect(toolRects[Tpat]);
        PenNormal;
        DisplayColors;
    end;
    with vSizeRect, UT[0] do
        SetRect(r, left, top+(vSize-1)*2, right, top+vSize*2);
        InvertRect(r);
    with hSizeRect, UT[0] do
        SetRect(r, left, top+(hSize-1)*2, right, top+hSize*2);
        InvertRect(r);
    (draw current tools, modes, pattern/color, pensize)
end;

Procedure OpenBitMap(r:rect); { create offscreen bitmap }
var
    pb      :Bitmap;
    xx,yy,sn :LongInt;
    i        :Integer;
    saveGDevice :GDHandle;
    maxGDevice  :GDHandle;
    theDepth    :Integer;
    offRowBytes :LongInt;
    sizeOfOff   :LongInt;
begin
    if MacII then
    begin
        if PixDraw then
        begin
            { * Pix Map Mak'in * }
            saveGDevice:=GetGDevice;
            maxGDevice:=GetMaxDevice(screenBits.Bounds);
            SetGDevice(maxGDevice);
            myCGrafPtr:=@myCGrafPort;
            OpenCPort(myCGrafPtr);
            theDepth:=myCGrafPtr^.portPixMap^.pixelSize;
            with r do
            begin
                offRowBytes:=((((theDepth*(right-left))+15))DIV 16)*2;
                sizeOfOff:=LongInt(bottom-top)*offRowBytes;
            end;
            with myCGrafPtr^.portPixMap^^ do
            begin
                DisposPtr(baseAddr);
                baseAddr:=NewPtr(sizeOfOff);
                rowBytes:=offRowBytes+$8000;
                bounds:=r;
            end;
            ourCMHandle:=maxGDevice^^.gdpMap^^.pmTable;
            err:=HandToHand(handle(ourCMHandle));
            If Err<>NoErr then
                SysBeep(1);
            ($R-)
            with ourCMHandle^^ do
            begin
                for i:=0 to ctSize do
                    ctTable[i].value:=i;

```



```

        transIndex:=BAND(transIndex,$7FFF);
    end;
    ($R+)
    myCGrafPtr^.portPixMap^.pmTable:=ourCMHandle;
    bmap:=BitMapPtr(myCGrafPtr^.portPixMap^);
    Wbits:=BitMapPtr(CGrafPtr(DrawWindow^.portPixMap^);
    SetPort(GrafPtr(myCGrafPtr));
    EraseRect(thePort^.portRect);
    SetPort(DrawWindow);
    SetGDevice(saveGDevice);
end;
else
begin
    with r do
    begin
        xx := right-left;
        yy := bottom-top;
    end;
    sn := ((xx+15) div 16)*2*yy;
    with bmap do
    begin
        bounds := r;
        rowBytes := ((xx+15) div 16)*2;
        BaseAddr := NewPtr(sn);
    end;
    pb := DrawWindow^.portbits;
    SetPortBits(bmap);
    ClipRect(r);
    RectRgn(DrawWindow^.visRgn,r);
    EraseRect(r);
    SetPortBits(pb);
    ClipRect(DrawWindow^.portRect);
    RectRgn(DrawWindow^.visRgn,DrawWindow^.portRect);
    Wbits:=DrawWindow^.portbits;
end;
end;

Procedure OpenDrawWindow; ( initialize drawing window )
var
    r :rect;
begin
    if MacII then
        DrawWindow := GetNewCWindow(129,nil,pointer(-1))
    else
        DrawWindow := GetNewWindow(129,nil,pointer(-1));
    SetPort(DrawWindow);
    dRect := DrawWindow^.portrect; ( drawing area )
    drect.left := dRect.left + 46; (make room for palette)
    setRect(r,0,0,576,720);
    OpenBitMap(r);
    prect := palette^.picFrame;
    OffSetRect(prect,-prect.left,-prect.top);
    ( palette bounds )
    DrawPalette;
end;

Procedure ChangeTool(NewTool:integer); ( set a new tool )
begin
    with LAPwBuf do
    begin
        theType := setTool;
        tool := NewTool;
    end;
    SendLap(255);
end;

Function DoSizeTool(pt:point;r:rect;cur:integer):integer;
var
    x :integer;
    r2 :rect;
begin
    repeat
        x := (pt.v-r.top) div 2;

```

```

    if x<1 then
        x := 1
    else if x>16 then
        x := 16;
    if x<cur then
        begin
            SetRect(r2,r.left,r.top+(cur-1)*2,r.right,r.top+cur*2);
            InvertRect(r2);
            cur := x;
            SetRect(r2,r.left,r.top+(cur-1)*2,r.right,r.top+cur*2);
            InvertRect(r2);
        end;
        GetMouse(pt);
        until MyGetNextEvent(mUpMask,myEvent);
        DoSizeTool := cur;
    end;

```

procedure PatternEditor(VAR myPat:Pattern);

```

var
    dlg : DialogPtr;
    aPat : Pattern;
    myEvent : EventRecord;
    done : Boolean;
    draw : Boolean;
    dh,dv : Integer;
    patRect : Rect;
    patEdit : Rect;
    tRect : Rect;
    savePort : GrafPtr;

function GetBitRect(index:Integer):Rect;
begin
    dh:=(index mod 8)*10;
    dv:=(index Div 8)*10;
    with tRect do
    begin
        left:=patEdit.left+dh;
        right:=left+10;
        top:=patEdit.top+dv;
        bottom:=top+10;
    end;
    InsetRect(tRect, 1,1);
    GetBitRect:=tRect;
end;

```

```

procedure InitPatBits;
var
    i : Integer;
begin
    for i:=0 to 63 do
        if BitTst(@aPat,i)
        then
            FillRect(GetBitRect(i),Black)
        else
            FillRect(GetBitRect(i),White);
    end;

```

```

function GetBitPos(pt:point):integer;
begin
    dh:=pt.h-patEdit.left;
    dh:=dh div 10;
    dv:=pt.v-patEdit.top;
    dv:=dv div 10;
    GetBitPos:=dv*8+dh;
end;

```

```

procedure EditPatClick(pt:point;firstOne:Boolean);
var
    index : integer;
begin
    index:=GetBitPos(pt);
    if firstOne then
        begin
            if BitTst(@aPat,index)

```

```

        then
            draw:=false
        else
            draw:=true;
    end;
    if draw
    then
        begin
            FillRect(GetBitRect(index),Black);
            BitSet(@aPat,index);
        end
    else
        begin
            FillRect(GetBitRect(index),White);
            BitClr(@aPat,index);
        end;
    FillRect(patRect,aPat);
end;

begin
    GetPort(savePort);
    aPat:=myPat;
    dlg:=GetNewDialog(1000,Nil,Pointer(-1));
    HiliteButton(dlg);
    done:=false;
    GetDItem(dlg,3,iType,iTem,patEdit);
    tRect:=patEdit;
    InsetRect(tRect,-1,-1);
    FrameRect(tRect);
    GetDItem(dlg,4,iType,iTem,patRect);
    tRect:=patRect;
    InsetRect(tRect,-1,-1);
    FrameRect(tRect);
    FillRect(patRect,aPat);
    InitPatBits;
    Repeat
        CheckQueue;
        If MyGetNextEvent(everyEvent,myEvent) then
            If isDialogEvent(myEvent) then
                if DialogSelect(myEvent,myWindow,itemHit) then
                    Case itemHit of
                        Ok,Cancel:Done:=True;
                        3:
                            with myEvent do
                                begin
                                    GlobalToLocal(where);
                                    EditPatClick(where,true);
                                    Repeat
                                        GetMouse(where);
                                        if PtInRect(where,patEdit) then
                                            EditPatClick(where,false);
                                        CheckQueue;
                                    Until Not StillDown;
                                end;
                            end;
                    end;
                (Case)
            Until done;
            if ItemHit=Ok then
                myPat:=aPat;
                DisposDialog(dlg);
                SetPort(savePort);
            end;
end;

Procedure SetupCursor;
var
    savebits,bm :bitmap;
    r            :rect;
begin
    with UT[0] do
        case theTool of
            Tbrush: if MacII then
                begin
                    SetCursor(arrow); ( * For now, manipulate color cursor * )
                end
            else

```

```

begin
    SetRect(bm.bounds,0,0,16,16);
    bm.baseAddr := @theCurs;
    bm.rowBytes := 2;
    saveBits := drawWindow^.portbits;
    SetPortBits(bm);
    EraseRect(bm.bounds);
    with UT[0] do
        SetRect(r,0,0,hSize,vSize);
        FillRect(r,black);
        SetPortBits(saveBits);
        SetCursor(theCurs);
    end;
    TSpatter: SetCursor(theSprayer^^);
    Tletters: SetCursor(GetCursor(iBeamCursor)^^);
    Terase: if MacII then
        begin
            SetCursor(arrow); ( * For now, manipulate color cursor * )
        end
    else
        begin
            SetRect(bm.bounds,0,0,16,16);
            bm.baseAddr := @theECurs;
            bm.rowBytes := 2;
            saveBits := drawWindow^.portbits;
            SetPortBits(bm);
            FillRect(bm.bounds,white);
            with UT[0] do
                SetRect(r,0,0,hSize,vSize);
                FrameRect(r);
                bm.baseAddr := @theECurs.mask;
                EraseRect(bm.bounds);
                FillRect(r,black);
                SetPortBits(saveBits);
                SetCursor(theECurs);
            end;
        end;
    Trect..Tfoval:
        SetCursor(GetCursor(crossCursor)^^);
    end;
    updateCurs := false;
end;

Procedure FixCursor; ( handle cursor updating )
var
    pt :point;
begin
    If windowPeek(FrontWindow^.windowKind>-1) then
        begin
            GetMouse(pt);
            if (PtInRect(pt,dRect) and arrowCurs) or updateCurs then
                begin
                    SetupCursor;
                    arrowCurs := false;
                end
            else if (not PtInRect(pt,dRect)) and (not
                arrowCurs) then
                begin
                    SetCursor(arrow);
                    arrowCurs := true;
                end;
        end;
end;

end;

Procedure SavePic(saveas:boolean); ( save file )
var
    where :point;
    reply :SFRReply;
    err :integer;
    ar,dr :packed array [0..75] of byte;
    n,l :longint;
    i,j,ref :integer;
    r,rr :rect;
    bm :bitmap;
    srcPtr :ptr;

```

```

dstPtr :ptr;
h       :handle;
s       :str255;

procedure ChkErr;
begin
    if err <> 0 then
        DoErr(err);
end;

begin
    if not saved then
        saves := true;
    if saves then
        begin
            SetPt(where,50,50);
            GetWTitle(drawWindow,s);
            SFPutFile(where,'Save document as:',s,@idleFilter,reply);
            BeginUpdate(drawWindow);
            DrawContents;
            DrawPalette;
            EndUpdate(drawWindow);
        end
    else
        with reply do
            begin
                good := true;
                GetWTitle(drawWindow,s);
                fname := s;
                vRefNum := fVref;
            end;
            if reply.good then
                begin
                    err := SetVol(nil,reply.vRefNum);
                    ChkErr;
                    if saves then
                        begin
                            err := FSDelete(reply.fname,reply.vRefNum);
                            if (err<>noErr) and (err<>fnfErr) then
                                ChkErr;
                            err :=
                                Create(reply.fname,reply.vRefNum,'JPNT','PNTG');
                                ChkErr;
                        end;
                            err := FSOpen(reply.fname,reply.vrefnum,ref);
                            ChkErr;
                            if saves then
                                begin
                                    err := SetEOF(ref,512);
                                    ChkErr;
                                    err := SetFPos(ref,fsFromStart,0);
                                    ChkErr;
                                    l := 0;
                                    n := 4;
                                    err := FSWrite(ref,n,@1);(write version # 0)
                                    ChkErr;
                                end;
                                    err := SetFPos(ref,fsFromStart,512);
                                    ChkErr;
                                    with bm do
                                        begin
                                            SetRect(bounds,0,0,72*8,1);
                                            baseAddr := @ar;
                                            rowbytes := 72;
                                        end;
                                        SetRect(rr,0,0,576,1);
                                        for i := 0 to 719 do
                                            begin
                                                setRect(r,0,i,576,i+1);
                                                CopyBits(bmap,bm,r,rr,SrcCopy,nil);
                                                srcPtr := @ar;
                                                dstPtr := @dr;
                                                PackBits(srcPtr,dstPtr,72);
                                                n := ord(dstPtr)-ord(@dr);

```

```

err := FSWrite(ref,n,@dr);
ChkErr;
CheckQueue;
end;
err := FSClose(ref);
ChkErr;
SetWTitle(drawWindow,reply.fname);
fVref := reply.vRefnum;
changed := false;
saved := true;
end;
end;

Function MySaveProc(dlg:DialogPtr;Var
theEvent:EventRecord;VAR IH:Integer):Boolean;
begin
    SetUpA5;
    MySaveProc:=false;
    CheckQueue;
    RestoreA5;
end;

Function Continue:Boolean;
var
    choice : Integer;
begin
    Continue:=true;
    if changed then
        begin
            choice:=NoteAlert(2000,@MySaveProc);
            Case Choice of
                2:begin
                    SavePic(false);
                    if changed then
                        Continue:=false;
                    end;
                3:Continue:=false;
            end;
        end;
    end;
end;

Procedure LoadPic; ( load picture )
var
    where :point;
    reply :SFReply;
    err :integer;
    ar :packed array [0..75] of byte;
    n :longint;
    i,j,ref :integer;
    r,rr :rect;
    bm :bitmap;
    srcPtr,p :ptr;
    dstPtr :ptr;
    typeList :SFTYPEList;

procedure ChkErr;
begin
    if err <> 0 then
        DoErr(err);
end;

begin
    if not Continue then
        exit(LoadPic);
    SetPt(where,80,50);
    typeList[0] := 'PNTG';
    SFFGetFile(where,'Open
document:',nil,1,typeList,@idleFilter,reply);
    if reply.good then
        begin
            BeginUpdate(drawWindow);
            DrawContents;
            DrawPalette;
            EndUpdate(drawWindow);

```

```

err := SetVol(nil,reply.vRefNum);
ChkErr;
err := FSOpen(reply.fname,reply.vrefnum,ref);
ChkErr;
err := SetFPos(ref,fsFromStart,512);
ChkErr;
with bm do
begin
SetRect(bounds,0,0,72*8,1);
baseAddr := @ar;
rowbytes := 72;
end;
err := GetEOF(ref,n);
ChkErr;
n := n-512;
p := NewPtr(n);
srcPtr := p;
err := FSRead(ref,n,srcPtr);
ChkErr;
SetRect(rr,0,0,576,1);
for i := 0 to 719 do
begin
dstPtr := @ar;
UnPackBits(srcPtr,dstPtr,72);
setRect(r,0,i,576,i+1);
CopyBits(bm,bmap,rr,r,SrcCopy,nil);
CheckQueue;
end;
disposPtr(p);
err := FSClose(ref);
ChkErr;
SetWTitle(drawWindow,reply.fname);
fVref := reply.vRefnum;
changed := false;
saved := true;
InvalRect(direct);
end;
end;

Procedure FadeIn(r:rect;pat:pattern);
var
v,h,x,i,n : longint;
begin
pennormal;
penpat(pat);
with r do
begin
v := bottom-top;
h := right-left;
for x := 1 to h do
begin
n := (x*v) div h;
for i := 1 to n do
begin
moveto(x+left,top+(i*v) div n);
line(0,0);
end;
end;
end;
end;
end;

Procedure FadeOut(r:rect;pat:pattern);
var
v,h,x,i,n : longint;
begin
pennormal;
penpat(pat);
with r do
begin
v := bottom-top;
h := right-left;
for x := 1 to h do
begin
n := ((h-x)*v) div h;

```

```

for i := 1 to n do
begin
moveto(x+left,top+(i*v) div n);
line(0,0);
end;
end;
end;
end;

Procedure FadeInto(r:rect;inPat,outPat:pattern);
begin
SysBeep(1);
end;

Procedure PastePicture;
var
PPHd1 : Handle; (* Paste Picture handle *)
PTHd1 : Handle; (* Paste TEXT handle *)
ScLen : Longint;
offset : Longint;
anEvent : EventRecord;
PicRect : Rect;
err : Longint;
info : FontInfo;
TEXTStr : Str255;
TBox : Rect;
begin
err:=LoadScrap;
PPHd1:=NewHandle(0);
ScLen:=GetScrap(PPHd1,'PICT',offset);
if ScLen<0 then
begin
PTHd1:=NewHandle(0);
ScLen:=GetScrap(PTHd1,'TEXT',offset);
if ScLen>-1 then
begin
GetFontInfo(info);
GetText(PTHd1,TEXTStr);
with DrawWindow^.portRect, info do
SetRect(TBox,left,top,left+StringWidth(TEXTStr)+1,top+ascent+descent+leading);
HLock(PTHd1);
PPHd1:=Handle(OpenPicture(TBox));
TextBox(PTHd1,ScLen,TBox,teJustLeft);
ClosePicture;
HunLock(PTHd1);
end;
DisposHandle(PTHd1);
end;
if ScLen>-1 then
begin
SetCursor(ThePlacer^^);
Repeat
CheckQueue;
until MyGetNextEvent(mDownmask,anEvent);
with anEvent do
begin
GlobalToLocal(where);
if (what=mouseDown) &
(PtinRect(where,DrawWindow^.portRect) & not
(PtinRect(where,prect))) then
with PicHandle(PPHd1)^^.picFrame,where do
begin
ClipRect(dRect);
SetRect(PicRect,h,v,h+(right-left),v+(bottom-top));
DrawPicture(PicHandle(PPHd1),PicRect);
OffBits;
DrawPicture(PicHandle(PPHd1),PicRect);
OnBits;
ClipRect(DrawWindow^.portRect);
end;
end;
updateCurs:=true;
end;

```

```

    DisposHandle(PPHdl);
end;

Procedure ResizeSprayKan;
var
    ResizeBox : Rect;
    CircleBox : Rect;
    savePort : GrafPtr;
    pt : Point;
    tStr : Str255;
    x,y : Integer;
    newRad : Integer;
    newSpeed : Integer;
    sr : Integer; ( * Save radius * )

procedure ShowSize(where:Point);
var
    dx,dy,dd : Integer;
begin
    with where do
        begin
            dx:=abs(x-h);
            dy:=abs(y-v);
        end;
        If dx>dy
        then
            dd:=dx
        else
            dd:=dy;
        If dd<sr then
            EraseRect(ResizeBox);
            sr:=dd;
            SetRect(CircleBox,x-dd,y-dd,x+dd,y+dd);
            FillOval(CircleBox,Black);
            newRad:=dd;
        end;
begin
    GetPort(savePort);
    Dlg:=GetNewDialog(1001,Nil,Pointer(-1));
    HiliteButton(Dlg);
    SetCursor(theSizer^^);
    newRad:=UT[0].splatRad;
    PenNormal;
    GetDItem(Dlg,5,iType,iTem,Box);
    newSpeed:=UT[0].splatSpeed;
    SetiText(iTem,Str(newSpeed));
    SeliText(Dlg,5,0,255);
    GetDItem(Dlg,3,iType,iTem,Box);
    ResizeBox:=Box;
    InsetRect(Box,-1,-1);
    FrameRect(Box);
    sr:=newRad;
    with ResizeBox do
        begin
            x:=left+(right-left) Div 2;
            y:=top+(bottom-top) Div 2;
            SetRect(CircleBox,x-sr,y-sr,x+sr,y+sr);
        end;
        FillOval(CircleBox,Black);
        ItemHit:=0;
    Repeat
        CheckQueue;
        If MyGetNextEvent(everyEvent,myEvent) then
            If isDialogEvent(myEvent) then
                if DialogSelect(myEvent,Dlg,itemHit) then
                    Case itemHit of
                        ok: begin
                            GetDItem(Dlg,5,iType,iTem,Box);
                            GetiText(iTem,tStr);
                            newSpeed:=Val(tStr);
                            If newSpeed<1
                                then
                                    begin

```

```

                                SysBeep(1);
                                newSpeed:=1;
                            end
                        else if newSpeed>255 then
                            begin
                                SysBeep(1);
                                newSpeed:=255;
                            end;
                    end;
                3: with myEvent do
                    begin
                        GlobalToLocal(where);
                        pt:=where;
                        ShowSize(where);
                        Repeat
                            GetMouse(where);
                            If PtInRect(where,ResizeBox) then
                                If longInt(where)<>longInt(pt) then
                                    begin
                                        ShowSize(where);
                                        pt:=where;
                                    end;
                                CheckQueue;
                                Until Not StillDown;
                            end;
                        end; ( case )
                    Until ItemHit in [ok,cancel];
                    If ItemHit=ok then
                        begin
                            with LAPwbuff do
                                begin
                                    theType := setSplat;
                                    sRad := newRad;
                                    sSpeed := newSpeed;
                                end;
                                SendLAP(255);
                            end;
                            DisposDialog(Dlg);
                            SetPort(savePort);
                        end;

Procedure ToolClick(pt:point);
var
    i,a,h,v : integer;
    dblClick : boolean;
    where : point;
    outColor : RGBColor;
begin
    dblClick := (TickCount-clickTime<GetDb1Time); (
possible double click? )
    clickTime := TickCount;
    a := 0;
    for i := 1 to 20 do
        if ptInRect(pt,toolRects[i]) then
            begin
                a := i;
                leave;
            end;
    dblClick:=(a=lastTool) and dblClick;
    if a>0 then
        begin
            lastTool := a;
            case a of
                1: begin ( spray can )
                    with LAPwbuff do
                        begin
                            theType := paintMode;
                            mode := not UT[0].theMode;
                        end;
                        SendLAP(255);
                        PenMode(PatXOR);
                        PenPat(1tgray);
                        PaintRect(toolRects[1spray]);
                        PenNormal;

```

```

end;
2..9: if dblClick then
  case a of
    TEraser: begin
      LAPwbuf.theType := eraseall;
      SendLAP(255);
    end;
    TSpatter:
      ResizeSprayKan;
  end
else
  with UT[0] do { Handle new tools }
    if theTool <> a then
      begin
        InvertRect(ToolRects[theTool]);
        InvertRect(ToolRects[a]);
        ChangeTool(a);
      end;
    10: begin { disk icon }
      end;
    11,12: if (PatternsUp <> (a=11)) and MacII then
      ( pattern/color switch )
      begin
        PenMode(PatXOR);
        PenPat(1tgray);
        PaintRect(toolRects[tpat]);
        PaintRect(toolRects[tcolor]);
        PenNormal;
        PatternsUp := not PatternsUp;
        if PatternsUp then
          DisplayPatterns
        else
          DisplayColors;
        end
      else if PatternsUp <> (a=11) then
        sysbeep(1);
    13..20: if dblClick then
      begin
        if patternsUp then
          begin
            PatternEditor(thePatterns[a-12]);
            with LAPwbuf do
              begin
                theType := setpat;
                pat := thePatterns[a-12];
              end;
            SendLAP(255);
            curPat := a-12;
            DisplayPatterns;
          end
        else
          begin
            { color picker }
            SetPt(where,0,0); (* The
color picker will center itself *)
            if GetColor(where,'Set Pallete
color to:',theColors[a-12],outColor) then
              begin
                theColors[a-12]:=outColor;
                with LAPwbuf do
                  begin
                    theType := setcolor;
                    clr := outColor;
                  end;
                SendLAP(255);
              end;
            curColor := a-12;
            DisplayColors;
          end;
        end
      else { single click }
      begin
        if PatternsUp and (curPat <> a-12) then
          ( pattern/color selection )

```

```

      begin
        with LAPwbuf do
          begin
            theType := setpat;
            pat := thePatterns[a-12];
          end;
        SendLAP(255);
        curPat := a-12;
        FillRect(curPatRect,thePatterns[curPat]);
      end
    else if not(patternsUp) and
(curColor <> a-12) then
      begin
        with LAPwbuf do
          begin
            theType := setcolor;
            clr := theColors[a-12];
          end;
        SendLAP(255);
        curColor := a-12;
        RGBForeColor(theColors[curColor]);
        PaintRect(curPatRect);
      end;
    end;
    updateCurs := true;
  end
else if PtInRect(pt,hSizeRect) then
  begin
    v := UT[0].vSize;
    h := DoSizeTool(pt,hSizeRect,UT[0].hSize);
    with LAPwbuf do
      begin
        theType := setPen;
        px := h;
        py := v;
      end;
    SendLAP(255);
    updateCurs:=true;
  end
else if PtInRect(pt,vSizeRect) then
  begin
    h := UT[0].hSize;
    v := DoSizeTool(pt,vSizeRect,UT[0].vSize);
    with LAPwbuf do
      begin
        theType := setPen;
        px := h;
        py := v;
      end;
    SendLAP(255);
    updateCurs := true;
  end;
end;

Function Sgn(i:integer):integer;
begin
  if i<0 then
    sgn := -1
  else if i>0 then
    sgn := 1
  else
    sgn := 0;
  end;
end;

Procedure MakeRect(pt1:point);
var
  LastPt,pt :point;
  x,y :integer;
  dx,dy :integer;
  r :rect;
begin
  lastPt := pt1;
  with pt1 do

```

```

begin
  x := h;
  y := v;
end;
Repeat
  GetMouse(Pt);
  if pt.h>direct.right then (pin mouse inside direct)
    pt.h := direct.right
  else if pt.h<direct.left then
    pt.h := direct.left;
  if pt.v>direct.bottom then
    pt.v := direct.bottom
  else if pt.v<direct.top then
    pt.v := direct.top;
  if longint>LastPt<>longint(pt) then ( mouse
moved ) ( NOTE: this is faster than EqualPt )
  begin
    ( - erase old rect - )
    SetRect(r,x,y>LastPt.h>LastPt.v);
    if x>LastPt.h then
      begin
        r.left := LastPt.h;
        r.right := x;
      end;
    if y>LastPt.v then
      begin
        r.top := LastPt.v;
        r.bottom := y;
      end;
    CopyBits(bmap,wbits,r,r,srcCopy,nil); (
flicker-matic quick fix )
    ( - check for shift constraint - )
    if bitAnd(mods,shiftKey)<>0 then
      begin
        dx := abs(pt.h-pt1.h);
        dy := abs(pt.v-pt1.v);
        if dy<dx then
          pt.h := pt1.h + dy*sgn(pt.h-pt1.h)
        else
          pt.v := pt1.v + dx*sgn(pt.v-pt1.v);
        end;
        ( - calculate new rect - )
        SetRect(r,x,y,pt.h,pt.v);
        if x>pt.h then
          begin
            r.left := pt.h;
            r.right := x;
          end;
        if y>pt.v then
          begin
            r.top := pt.v;
            r.bottom := y;
          end;
        ( - fix mistakes - )
        with UT[0] do
          begin
            PenPat(thePat);
            if MacII then
              RGBForeColor(theClr);
            PenSize(hSize,vSize);
          end;
          ( - draw rect - )
          case UT[0].theTool of
            TRect: begin
              if bitAnd(mods,optionKey)=0 then
                PenPat(black);
                FrameRect(r);
              end;
            TFRect: begin
              PaintRect(r);
              if bitAnd(mods,optionKey)=0 then
                begin
                  PenPat(black);

```

```

                FrameRect(r);
              end;
            end;
          T0val: begin
            if bitAnd(mods,optionKey)=0 then
              PenPat(black);
              FrameOval(r);
            end;
          TF0val: begin
            PaintOval(r);
            if bitAnd(mods,optionKey)=0 then
              begin
                PenPat(black);
                FrameOval(r);
              end;
            end;
          end;
        end;
        LastPt := pt;
        CheckQueue;
        Until MyGetNextEvent(mupmask,myEvent);
        with LAPwbuf do
          begin
            theType := rectpck; ( send rectpck packet )
            rct := r;
            optDown := (bitAnd(mods,optionKey)<>0);
          end;
          SendLAP(255); ( send message to everyone )
        end;
      Procedure MasterClick(pt:Point);(mouseDown our window)
      var
        LastPt :Point;
        dx,dy :integer;
        mypart :integer;
      begin
        ( * If a DA then get of of here * )
        if windowPeek(FrontWindow)~.windowKind<0 then
          exit(MasterClick);
        if PtInRect(pt,rect) then ( click in palette )
          ToolClick(pt)
        else
          begin
            if UT[0].theTool in [TRect..TF0val] then
              MakeRect(pt)
            else
              begin
                with LAPwbuf do
                  begin
                    theType := setpos; ( send setpos packet )
                    mx := pt.h;
                    my := pt.v;
                  end;
                  SendLAP(255); (send setpos packet to everyone)
                Repeat
                  GetMouse(Pt);
                  if pt.h>direct.right then ( pin
mouse inside direct )
                    pt.h := direct.right
                  else if pt.h<direct.left then
                    pt.h := direct.left;
                  if pt.v>direct.bottom then
                    pt.v := direct.bottom
                  else if pt.v<direct.top then
                    pt.v := direct.top;
                  if (longint>LastPt<>longint(Pt)) |
(UT[0].TheTool=Tsplatter) then ( mouse moved ) ( NOTE: this
is faster than EqualPt )
                    begin
                      with LAPwbuf do
                        begin
                          theType := drag; (send drag packet)
                          cx := pt.h;
                          cy := pt.v;

```

```

        end;
        SendLAP(255); (send message to everyone)
        LastPt := Pt;
    end;
    CheckQueue;
    Until MyGetNextEvent(mupmask,myevent);
end;
end;
end;

Procedure MasterKey(theChar:char);
begin
    if UT[0].theTool = Tletters then
        if theChar>=' ' then
            begin
                with LAPwbuf do
                    begin
                        theType := alpha;
                        ch := theChar;
                    end;
                    SendLAP(255);
                end;
            end;
end;

Procedure ChangeFont;
var
    s :str255;
    i :integer;
begin
    CheckItem(myMenus[fontMenu],theFontidx,false);
    theFontidx:=theItem;
    GetItem(myMenus[fontMenu],theItem,s);
    GetFNum(s,theItem);
    CheckItem(myMenus[fontMenu],theFontidx,true);
    with UT[0] do
        if theItem<>theFNum then
            begin
                with LAPwbuf do
                    begin
                        theType := setfont;
                        fnum := theItem;
                        fsize := theFSize;
                        fstyl := theFstyle;
                    end;
                    SendLAP(255);
                end;
            end;
        for i:=1 to 9 do
            begin
                GetItem(myMenus[sizeMenu],i,s);
                if RealFont(theItem,Val(s))
                    then
                        SetItemStyle(myMenus[sizeMenu],i,[Outline])
                    else
                        SetItemStyle(myMenus[sizeMenu],i,[]);
            end;
        end;
end;

procedure ChangeStyle;
const
    plainItem = 1;
var
    markChar :char;
    StyleArray :packed array [1..7] of styleitem;
    i :integer;
    CStyle :style;
begin
    CStyle:=UT[0].theFStyle;
    StyleArray[1]:=Bold;
    StyleArray[2]:=Italic;
    StyleArray[3]:=Underline;
    StyleArray[4]:=Outline;
    StyleArray[5]:=Shadow;
    StyleArray[6]:=Extend;
    StyleArray[7]:=Condense;

```

```

if theitem=plainItem then
    begin
        CheckItem(myMenus[styleMenu],1,true);
        for i:=2 to 8 do
            CheckItem(myMenus[styleMenu],i,false);
            CStyle:=[];
        end
    else
        begin
            CheckItem(myMenus[styleMenu],1,false);
            GetItemMark(myMenus[styleMenu],theitem,markChar);
            if markChar=chr(noMark) then
                begin
                    CStyle:=CStyle+StyleArray[theitem-1];
                    CheckItem(myMenus[styleMenu],theitem,True);
                end
            else
                begin
                    CheckItem(myMenus[styleMenu],theitem,false);
                    CStyle:=CStyle-StyleArray[theitem-1];
                    if CStyle=[] then
                        CheckItem(myMenus[styleMenu],1,true);
                    end;
                end;
            end;
            if CStyle<>UT[0].theFStyle then
                begin
                    with LAPwbuf do
                        begin
                            theType := setfont;
                            with UT[0] do
                                begin
                                    fNum := theFNum;
                                    fSize := theFSize;
                                    fstyl := CStyle;
                                end;
                            end;
                            SendLAP(255);
                        end;
                    end;
                end;
            end;

Procedure ChangeSize;
var
    s :str255;
begin
    if theSizeidx<>theItem then
        begin
            CheckItem(myMenus[sizeMenu],theSizeidx,false);
            CheckItem(myMenus[sizeMenu],theItem,true);
            theSizeidx:=theItem;
            GetItem(myMenus[sizeMenu],theItem,s);
            theItem:=Val(s);
            with UT[0] do
                begin
                    with LAPwbuf do
                        begin
                            theType := setfont;
                            fnum := theFNum;
                            fsize := theItem;
                            fstyl := theFstyle;
                        end;
                        SendLAP(255);
                    end;
                end;
            end;
        end;
    end;

Procedure DrawContents; ( redraw drawing )
begin
    SetPort(DrawWindow);
    CopyBits(bmap,wbits,direct,direct,SrcCopy,Nil);
end;

Procedure PrintPic; ( print a document )
var
    GetOutEh : Boolean;

```



```

temphPrint : THPrint;
err        : OSErr;
savePort   : GrafPtr;
myPrPort   : TPrPort;
myStRec    : TPrStatus;
i          : Integer;
begin
    temphPrint:=hPrint;
    err:=HandToHand(handle(temphPrint));
    if Err<>NoErr then
        begin
            SysBeep(1);
            exit(PrintPic);
        end;
    PrOpen;
    if PrJobDialog(temphPrint)
    then
        begin
            GetOutEh:=false;
            DisposHandle(handle(hPrint));
            If MemError<>NoErr then
                SysBeep(1);
            hPrint:=temphPrint;
            err:=HandToHand(handle(hPrint));
            If Err<>NoErr then
                begin
                    SysBeep(1);
                    GetOutEh:=true;
                end;
            end
        else
            GetOutEh:=true;
        PrClose;
        DisposHandle(handle(temphPrint));
        if GetOutEh then
            exit(PrintPic);
        BeginUpdate(DrawWindow);
        DrawContents;
        DrawPalette;
        EndUpdate(DrawWindow);
        GetPort(savePort);
        for i:=1 to hPrint^.PrJob.iCopies do
            begin
                PrOpen;
                If PrError=noErr then
                    begin
                        myPrPort:=PrOpenDoc(hPrint,Nil,Nil);
                        If PrError=noErr then
                            begin
                                PrOpenPage(myPrPort,Nil);
                                If PrError=noErr then
                                    CopyBits(bmap,myPrPort^.gPort.portBits,bmap.bounds,bmap.bounds,SrcCopy,Nil);
                                    PrClosePage(myPrPort);
                                end;
                            end;
                        end;
                        If PrError=noErr then
                            PrCloseDoc(myPrPort);
                        If (hPrint^.PrJob.bjDocLoop=bSpoolLoop) and
                        (PrError=NoErr) then
                            PrPicFile(hPrint,Nil,Nil,Nil,myStRec);
                            PrClose;
                        end;
                        SetPort(savePort);
                    end;
                Procedure DoMyUpdate;
                var
                    savePort :grafPtr;
                    tempWindow :windowPtr;
                begin
                    tempWindow:=WindowPtr(myEvent.message);
                    GetPort(savePort);
                    SetPort(tempWindow);

```

```

                    BeginUpdate(tempWindow);
                    if tempWindow=DrawWindow then
                        begin
                            DrawContents;
                            DrawPalette;
                        end;
                    EndUpdate(tempWindow);
                    SetPort(savePort);
                end;
            Procedure DoJamAbout;
            const
                picID = 999;
            var
                AboutWindow: WindowPtr;
                AboutPic    : PicHandle;
                AboutRect    : Rect;
                AboutEvent   : EventRecord;
                SavePort     : GrafPtr;
                x1,y1,x2,y2: Integer;
            begin
                GetPort(savePort);
                AboutPic:=PicHandle(GetResource('PICT',999));
                RsrcErr;
                with ScreenBits.Bounds do
                    begin
                        x1:=right-left;
                        y1:=bottom-top;
                    end;
                with AboutPic^.picFrame do
                    begin
                        x2:=right-left;
                        y2:=bottom-top;
                    end;
                SetRect(AboutRect,1,1,x2,y2);
                AboutWindow:=NewWindow(Nil,AboutRect','',false,1,Pointer(-1),false,0);
                MoveWindow(AboutWindow,(x1-x2) Div 2,(y1-y2) Div 2,true);
                ShowWindow(AboutWindow);
                SetPort(AboutWindow);
                with AboutPic^.picFrame do
                    SetRect(AboutRect,0,0,right-left,bottom-top);
                DrawPicture(AboutPic,AboutRect);
                Repeat
                    CheckQueue;
                Until
                    MyGetNextEvent(mdownMask+keyDownMask+AutoKeyMask,AboutEvent);
                ReleaseResource(handle(AboutPic));
                RsrcErr;
                DisposeWindow(AboutWindow);
                SetPort(savePort);
            end;
            Procedure DoCommand(mResult:longint); {menu commands}
            var
                name : Str255;
            begin
                theMenu := HiWord(mResult);
                theItem := LoWord(mResult);
                if BitAnd(myEvent.modifiers,CmdKey) <> 0 then
                    HiliteMenu(theMenu);
                case theMenu of
                    255: If theItem <> 1 then
                        begin
                            GetItem(myMenus[appleMenu],theItem,name);
                            refnum:=OpenDeskAcc(name);
                        end
                    else If WindowPeek(FrontWindow)^.windowKind > 0 then
                        DoJamAbout
                    else
                        SysBeep(1);
                255: case theItem of
                    1: LoadPic;

```

```

2: SavePic(false);
3: SavePic(true);
( — )
5: DoPageSetUp;
6: PrintPic;
( — )
8: If Continue then
doneFlag:=true;
10:begin
PrDebug := not PrDebug;
if PrDebug then
begin
PrCtlCall(iPrDevCtl,$00010000,0,0);
debug('→ Debug transcript follows');
end;
end;
end;
257: ChangeFont;
258: ChangeSize;
259: ChangeStyle;
260: case theItem of
1: FadeIn(drawWindow^.portRect,UT[0].thePat);
2: FadeOut(drawWindow^.portRect,UT[0].thePat);
3: FadeInto(drawWindow^.portRect,black,white);
end;
end;
HiliteMenu(0);
end;

Procedure InitTables; ( initialize User Table, Queue )
var
i :integer;
begin
with PQ do ( empty queue )
begin
head := 0;
tail := 0;
end;
with UT[0] do
begin
id := myNode;
theTool := dTool; ( default tool )
theMode := dMode; ( default mode )
thePat := black; ( default pattern )
theClr := theColors[dColor]; ( default to black )
hSize := dPen; ( default pensize )
vSize := dPen; ( default pensize )
theFnum := applFont; ( default text font )
theFsize := dSize; ( default text size )
theFstyle := dStyle; ( default text style )
x := 100; ( somewhere on the screen )
y := 50;
splatSpeed := dsplatterCount; ( default speed )
splatRad := dsplatRad; ( default radius )
( time needs no initialization )
end;
for i := 1 to maxUsers do
begin
UT[i] := UT[0];
UT[i].id := -1; ( invalidate all users )
end;
end;

Procedure ReadPatCol; (read patterns & colors into memory)
var
RGBColors :CHandle;
i :integer;
begin
for i:=1 to 8 do
GetIndPattern(thePatterns[i],128,i);
RGBColors:=CHandle(GetResource('CLRS',128));
RsrcErr;
theColors:=RGBColors^^;
ReleaseResource(handle(RGBColors));

```

```

RsrcErr;
end;

Procedure InitGlob; ( init globals, user table, etc. )
const
UnimplTrapNum = $9F; ( * Unimplemented trap * )
WaitNextEventTrapWord = $60;
var
i,j :integer;
Rom,Machine :integer;
theWorld :SysEnvRec;
begin
( * Get the world, so to speak * )
if SysEnvirons(1,theWorld)<>envNotPresent then
begin
MultiFinderRunning:=(theWorld.machineType=0) &
( GetTrapAddress(WaitNextEventTrapWord)<>GetTrapAddress(UnimplTrapNum,ToolTrap));
ColorQDrawImplm:=theWorld.hasColorQD;
end
else
begin
MultiFinderRunning:=false;
ColorQDrawImplm:=false;
end;
end;
Environs(Rom,Machine);
if Machine=2
then
MacII:=true
else
MacII:=false;
PixDraw:=false;
( * Controls if mac II pix map is working or not * )
MacII:=false;
PrDrvOpen;
PrDebug := false; ( *** for debugging only *** )
err := MPPOpen; ( open Appletalk driver )
if err<>noErr then
begin
SysBeep(1);
exitToShell;
end;
err := GetNodeAddress(myNode,myNet); ( who am I ? )
if err<>noErr then
begin
SysBeep(1);
exitToShell;
end;
err := LapOpenProtocol(OurType,nil); ( open our
protocol type )
if err<>noErr then
begin
SysBeep(1);
exitToShell;
end;
LAPrh := ABReCHandle(newHandle(lapSize)); ( handle for
LAP reads )
LAPwh := ABReCHandle(newHandle(lapSize)); ( handle for
LAP writes )
HLock(handle(LAPrh));
HLock(handle(LAPwh));
hPrint:=THPrint(NewHandle(SizeOf(TPrint)));
SetUpRead; ( set up initial LAPRead )
palette := PicHandle(GetResource('PICT',1000));
RsrcErr;
jamPic := PicHandle(GetResource('PICT',1001));
RsrcErr;
for i := 1 to 20 do
begin
j := (i-1) div 2;
if odd(i) then
SetRect(ToolRects[i],0,20*j,22,20*(j+1)-1)
else
SetRect(ToolRects[i],23,20*j,45,20*(j+1)-1);
end;
end;

```

```

SetRect(curPatRect,3,203,42,225);
SetRect(hSizeRect,0,231,21,266);
SetRect(vSizeRect,23,231,44,266);
PatternsUp := true;
curPat := 1;
curColor := 1;
ReadPatCol;
PatternsUp:=true;
for i := 0 to 15 do
    theCurs.mask[i] := 0;
setPt(theCurs.hotspot,0,0);
setPt(theECurs.hotspot,0,0);
arrowCurs := true;
updateCurs := false;
InitTables;
OpenDrawWindow;
textMode(SrcCopy);
clickTime := 0; ( set up doubleclick variables )
lastTool := -1;
DoneFlag := false;
theSizeidx := 3;
changed := false;
theSizer:=GetCursor(1000);
theHand:=GetCursor(1001);
thePlacer:=GetCursor(1002);
theSprayer:=GetCursor(1003);
end;

Procedure Cleanup;
( All of this isn't really necessary except for closing our
protocol type. The application heap will be flushed anyway,
but we ought to set a good example.... )
begin
    PrDrvrclose;
    if LAPrh^.abResult=1 then(stop LAP read if in on)
    begin
        err := LAPRdCancel(LAPrh);
        DoErr(1004);
    end;
    if not polling then
    begin
        err := VRemove(@myTask); ( remove VBLTask )
        DoErr(1005);
    end;
    err := LAPCcloseProtocol(OurType); (! close protocol !)
    DoErr(1000);
    err := MPPCclose; ( close AppleTalk driver )
    DoErr(1002);
    DisposHandle(handle(LAPrh));
    DisposHandle(handle(LAPwh));
    releaseResource(handle(palette));
    RsrcErr;
    releaseResource(handle(jamPic));
    RsrcErr;
    if PixDraw
    then
        begin
            begin
            end
        else
            DisposPtr(bmap.baseAddr);
        end
    end;

procedure _DataInit;EXTERNAL;
begin
    UnloadSeg(@_DataInit);(* Get rid of MPW's init code *)
    FlushEvents(everyEvent,0);
    InitGraf(@thePort);
    InitFonts;
    TEInit;
    InitWindows;
    InitDialogs(Nil);
    InitCursor;
    SetUpMenus;
    InitGlob;

```

```

Repeat
    FixCursor;
    CheckQueue; ( something for us to do while we idle )
    if MyGetNextEvent(everyEvent,myEvent) then
        with myEvent do
            case what of
                mouseDown:
                    begin
                        mods := modifiers;
                        code := FindWindow(where,tempWindow);
                        case code of
                            inMenuBar: DoCommand(MenuSelect(where));
                            InSysWindow: SystemClick(myEvent,tempWindow);
                            inContent,InDrag:
                                if tempWindow <> frontWindow then
                                    begin
                                        SelectWindow(tempWindow);
                                        SetPort(tempWindow);
                                    end
                                else if FrontWindow=DrawWindow then
                                    begin
                                        GlobalToLocal(where);
                                        MasterClick(where);
                                    end;
                                end;
                    end;
                keyDown, autoKey:
                    begin
                        theChar := chr(BitAnd(message,255));
                        if BitAnd(modifiers,CmdKey)<>0 then
                            begin
                                if theChar in ['v','V'] then
                                    PastePicture
                                else
                                    DoCommand(MenuKey(theChar))
                                end
                            end
                        else
                            MasterKey(theChar);
                        end;
                    updateEvt:
                        DoMyUpdate;
                    activateEvt:
                        if Odd(modifiers) then
                            begin
                                tempWindow:=WindowPtr(message);
                                SetPort(tempWindow);
                                if tempWindow=DrawWindow then
                                    begin
                                        updateCurs:=true;
                                        ( * Remove Edit menu/ we don't use it, but DAs might * )
                                        if FileMenuPresent then
                                            begin
                                                FileMenuPresent:=false;
                                                DeleteMenu(261);
                                                DrawMenuBar;
                                            end;
                                        end;
                                    end
                                else
                                    SetCursor(arrow);
                                end
                            end
                        else
                            begin
                                ( * If our about caused it then give them an Edit menu * )
                                if (WindowPtr(message)=DrawWindow) & (not
                                FileMenuPresent) & (windowPeek(FrontWindow)^.windowKind<0)
                                then
                                    begin
                                        FileMenuPresent:=true;
                                        InsertMenu(myMenus[EditMenu],257);
                                        DrawMenuBar;
                                    end;
                                end;
                            end;
                    end;
            until doneFlag;

```

```

CleanUp;
SetCursor(GetCursor(WatchCursor)^^);
end.

```

Listing: JamPaint.r

```

/* resources for JamPaint.p */
Type 'JpNT' as 'STR';
resource 'JpNT' (0) ("JamPaint by Edgar Circenis and Rod
Magnuson - Dec 28, 1987");
resource 'FREF' (128) ('APPL', 0, "");
resource 'BNDL' (128) (
    'JpNT', 0, ('ICN*', {0, 128}; 'FREF', {0, 128}));
resource 'MENU' (255, "Apple", preload) (
    255, textMenuProc, 0x7FFFFFFD, enabled, apple,
    ( "About JamPaint", noIcon, "", "", plain,
      "--", noIcon, "", "", plain));
resource 'MENU' (256, "File", preload) (
    256, 0, 0x7FFFFFFB7, enabled, "File",
    ( "Open...", noIcon, "O", noMark, plain;
      "Save", noIcon, "S", noMark, plain;
      "Save as...", noIcon, noKey, noMark, plain;
      "--", noIcon, noKey, noMark, plain;
      "Page Setup...", noIcon, noKey, noMark, plain;
      "Print...", noIcon, "P", noMark, plain;
      "--", noIcon, noKey, noMark, plain;
      "Quit", noIcon, "Q", noMark, plain));
resource 'MENU' (257, "Font", preload) (
    257, 0, 0x7FFFFFFF, enabled, "Font", {});
resource 'MENU' (258, "Size", preload) (
    258, textMenuProc, allEnabled, enabled, "Size",
    (
        "9", noIcon, "", "", plain,
        "10", noIcon, "", "", plain,
        "12", noIcon, "", check, plain,
        "14", noIcon, "", "", plain,
        "18", noIcon, "", "", plain,
        "24", noIcon, "", "", plain,
        "36", noIcon, "", "", plain,
        "48", noIcon, "", "", plain,
        "72", noIcon, "", "", plain));
resource 'MENU' (259, "Style", preload) (
    259, textMenuProc, allEnabled, enabled, "Style",
    (
        "Plain", noIcon, "", check, plain,
        "Bold", noIcon, "B", "", 1,
        "Italic", noIcon, "I", "", 2,
        "Underline", noIcon, "U", "", 4,
        "Outline", noIcon, "", "", 8,
        "Shadow", noIcon, "", "", 16,
        "Extend", noIcon, "", "", 64,
        "Condense", noIcon, "", "", 32));
resource 'MENU' (260, "Effects", preload) (
    260, textMenuProc, allEnabled, enabled, "Effects",
    ( "Fade In", noIcon, noKey, noMark, plain;
      "Fade Out", noIcon, noKey, noMark, plain;
      "Fade from/to", noIcon, noKey, noMark, plain));
resource 'MENU' (261, "Edit", preload) (
    261, textMenuProc, 0x7FFFFFFF, enabled, "Edit",
    ( "Can't Undo", noIcon, "Z", "", plain,
      "--", noIcon, "", "", plain,
      "Cut", noIcon, "X", "", plain,
      "Copy", noIcon, "C", "", plain,
      "Paste", noIcon, "V", "", plain,
      "Clear", noIcon, "", "", plain));
resource 'WIND' (129, "Paint Window", preload) (
    39, 10, 341, 502,
    noGrowDocProc, visible, noGoAway, 0x0, "Untitled");
resource 'ALRT' (2000) (
    78, 122, 234, 364, 2000,
    ( Cancel, visible, sound1,
      Cancel, visible, sound1,
      Cancel, visible, sound1,
      Cancel, visible, sound1));
resource 'DITL' (2000) (

```

```

    ( 112, 32, 132, 92),
      Button (enabled, "Discard"),
      (72, 32, 92, 92),
      Button (enabled, "Save"),
      (72, 136, 92, 196),
      Button (enabled, "Cancel"),
      (24, 96, 40, 200),
      StaticText (disabled, "Save changes?"));
resource 'DLOG' (1000, "Pattern Editor") (
    78, 148, 208, 374,
    dBoxProc, visible, noGoAway, 0x0, 1000, "");
resource 'DITL' (1000) (
    ( 104, 152, 124, 212),
      Button (enabled, "Ok"),
      (104, 80, 124, 140),
      Button (enabled, "Cancel"),
      (8, 24, 88, 104),
      UserItem (enabled),
      (8, 128, 88, 208), UserItem (disabled));
resource 'DLOG' (1001, "Resize Spray Kan") (
    46, 98, 266, 410,
    dBoxProc, visible, noGoAway, 0x0, 1001, "");
resource 'DITL' (1001, "Resize Spray Kan") (
    ( 176, 232, 196, 292),
      Button (enabled, "OK"),
      (144, 232, 164, 292),
      Button (enabled, "Cancel"),
      (16, 16, 208, 208),
      UserItem (enabled),
      (40, 224, 56, 296),
      StaticText (disabled, "Flow Rate"),
      (70, 234, 84, 288),
      EditText (enabled, ""));
resource 'DLOG' (1002, "Printing Error!") (
    36, 108, 80, 392,
    dBoxProc, visible, noGoAway, 0x0, 1002, "");
resource 'DITL' (1002, "Printing Error!") (
    ( 13, 37, 29, 253),
      StaticText (disabled, "An error while printing oc-
cured!"));
data 'CLRS' (128, "colors", preload) (
    $0000 0000 0000 /* BLACK */
    $FFFF FFFF FFFF /* WHITE */
    $FFFF 0000 0000 /* RED */
    $0000 FFFF 0000 /* GREEN */
    $0000 0000 FFFF /* BLUE */
    $FFFF FFFF 0000 /* YELLOW */
    $0000 FFFF FFFF /* TURQUOISE */
    $FFFF 0000 FFFF /* MAGENTA */
);
resource 'PAT#' (128, "Patterns", preload) (
    (
        /* [Black] */
        $FFFF FFFF FFFF FFFF,
        /* [Dk Gray] */
        $DDFF 77FF DDFF 77FF,
        /* [Med Gray] */
        $DD77 DD77 DD77 DD77,
        /* [Gray] */
        $AA55 AA55 AA55 AA55,
        /* [White] */
        $0000 0000 0000 0000,
        /* [Spot White] */
        $8000 0800 8000 0800,
        /* [More Spots White] */
        $8800 2200 8800 2200,
        /* [Still More Spots White] */
        $8822 8822 8822 8822,
        /* [And Still More Spots White] */
        $AA00 AA00 AA00 AA00
    )
);
include "JamPaint.rsrc"; /* additional resources */
include "JamPaint.code";

```



ABC's of Macintosh

A Window Menu



Clifford Story
Goleta, CA
Volume 5, Number 7

The Standard Window Menu

A Macintosh program with several windows really ought to have a Window menu. This menu is a list of all open application windows; choosing a window from the menu brings it to the front.

The problem for the programmer is menu maintenance. Every time the program opens a new window, the window's name has to be added to the Window menu, and, what is worse, every time a window is closed, its name must be removed from the menu. There is no ROM trap to delete a menu item in the 64K ROM.

Still uglier problems arise if the menu is supposed to be kept in order. Every time a window is selected, the window order changes, and the menu must be re-ordered.

One solution: each time the menu must be updated, delete the whole menu and recreate it from the window list. That isn't very pretty, but it would work.

The solution I present here is similar but less work: Draw the menu with an MDEF that reads the window list. This approach requires virtually no menu maintenance (only a call to CalcMenuSize when a window is opened or closed). As an extra, the menu includes a "Zoom Front Window" item.

Advertisement

This is my third article for MacTutor, and I've got a fourth and a fifth in the works. Maybe I'm becoming a regular? Anyway, I thought I might mention ways to respond to my stuff.

The cheapest way is by mail:

Clifford Story
Attic Software
P.O. Box 219
Goleta, California 93116

The fastest way is by modem. I run a BBS called THE ATTIC, phone number (805) 683-0322, hours 6:00 PM to 2:00 AM, Pacific time, daily. The board isn't very active, but I read and respond each morning.

```
*****
# Window.make
# MPW make file for
#   dynamic Window menu demo
# (c) 1988, by Clifford Story
#   & Attic Software
*****
# Compile the resources
*****
Window ff Window.make Window.r
Rez Window.r -append -o Window
*****
# Compile common declarations
*****
```

```
Common.p.o ff Window.make Common.p
Pascal Common.p
*****
# Compile and link the MDEF
*****
'Window Menu'.p.o f Window.make d
'Window Menu'.p Common.p.o
Pascal 'Window Menu'.p
Window ff Window.make 'Window Menu'.p.o
Link -m MENUDEF -w -t 'APPL' d
-c '????' d -rt MDEF=1001 d
-sn Main='Window Menu' d
'Window Menu'.p.o d
"(Libraries)"Interface.o d
"(Libraries)"Runtime.o d
"(PLibraries)"PasLib.o d
"(PLibraries)"SANELib.o d
-o Window
*****
# Compile and link the main program
*****
Window.p.o f Window.make d
Window.p Common.p.o
Pascal Window.p
Window ff Window.make Window.p.o
Link -w -t APPL -c '????' d
Window.p.o d
"(Libraries)"Interface.o d
"(Libraries)"Runtime.o d
"(PLibraries)"PasLib.o d
"(PLibraries)"SANELib.o d
-o Window
*****
```

The Program's Resources

The only feature that needs to be pointed out is the declaration of the Window menu. The "Zoom Front Window" item must be declared here, so MenuKey will be able to find its keyboard equivalent.

```
/******
Window.r
Resources for
dynamic Window menu demo.
(c) 1988, by Clifford Story
& Attic Software
*****
#include "types.r"
/******
Menu resources:
*****
resource 'MENU' (1001) (
1001,
textMenuProc,
$7FFFFFFB,
enabled,
apple,
( /* array: 3 elements */
/* [1] */
"About Window...", noicon,
"", "", plain,
/* [2] */
```

```

"About Attic Soft-
ware...",
noicon, "", "",
plain,
/* [3] */
"--", noIcon, "", "",
plain
);
resource 'MENU' (1002,
preload) (
1002,
textMenuProc,
0x7FFFFFFB,
enabled,
"File",
( /* array: 4 elements */
/* [1] */
"New", noIcon, "N", "",
plain,
/* [2] */
"Close", noIcon, "K",
"", plain,
/* [3] */
"--", noIcon, "", "",
plain,
/* [4] */
"Quit", noIcon, "Q", "",
plain
);
resource 'MENU' (1003,
preload) (
1003,
textMenuProc,
0x7FFFFFFD,
enabled,
"Edit",
( /* array: 6 elements */
/* [1] */
"Undo", noIcon, "Z", "",
plain,
/* [2] */
"--", noIcon, "", "",
plain,
/* [3] */
"Cut", noIcon, "X", "",
plain,
/* [4] */
"Copy", noIcon, "C", "",
plain,
/* [5] */
"Paste", noIcon, "V",
"", plain,
/* [6] */
"Clear", noIcon, "", "",
plain
);
resource 'MENU' (1004,
preload) (
1004,
1001,
0x7FFFFFFF,
enabled,
"Window",
( /* array: 2 elements */
/* [1] */
"Zoom Front Window",
noicon,
"W", "", plain,
/* [2] */
"--", noIcon, "", "",
plain
);

```

```

);
/*****
Picture resources
*****/
resource 'PICT' (1001,
>About",
purgeable) (1157,
(7, 7, 306, 497),
$1101 A000 82A0 008C"
$0100 0A00 0700 0701"
$3201 F10A 0000 0000"
$0000 0000 0B00 1B00"
$1B44 0009 0009 0130"
$01EF 0700 0200 0248"
$A100 9600 0606 0000"
$0002 03A1 009A 0008"
$FFFD 0000 00D4 0000"
$A000 9803 0003 0000"
$0A28 0128 0024 22A9"
$2031 3938 3820 6279"
$2043 6C69 6666 6F72"
$6420 5374 6F72 7920"
$616E 6420 4174 7469"
$6329 AC22 2053 6F66"
$7477 6172 652C 2050"
$2E4F 2E20 426F 7820"
$3231 392C 2047 6F6C"
$6574 612C 2043 29A6"
$1161 6C69 666F 726E"
$6961 2020 2039 3331"
$3136 A000 99A0 0097"
$A100 9600 0605 0000"
$0002 03A1 009A 0008"
$004C 0000 00D8 0000"
$A000 980D 000C 2800"
$5200 2222 5769 6E64"
$6F77 2064 656D 6F6E"
$7374 7261 7465 7320"
$6120 6479 6E61 6D69"
$6320 5769 6E64 29ED"
$1D6F 7720 6D65 6E75"
$2E20 2045 6163 6820"
$7469 6D65 2061 2077"
$696E 646F 770D A000"
$99A1 009A 0008 003C"
$0000 00D8 0000 A000"
$9828 0062 0022 2269"
$7320 6F70 656E 6564"
$206F 7220 636C 6F73"
$6564 2C20 6F72 2074"
$6865 2077 696E 646F"
$7729 D81C 206F 7264"
$6572 2063 6861 6E67"
$6573 2C20 7468 6520"
$6D65 6E75 2069 730D"
$A000 99A1 009A 0008"
$002C 0000 00D8 0000"
$A000 9828 0072 0022"
$2261 7574 6F6D 6174"
$6963 616C 6C79 2075"
$7064 6174 6564 2074"
$6F20 7265 666C 6563"
$7420 7429 D61E 6865"
$206E 6577 2077 696E"
$646F 7720 6F72 6465"
$722C 2077 6974 6820"
$7468 650D A000 99A1"
$009A 0008 001C 0000"
$00D8 0000 A000 9828"
$0082 0022 2274 6F70"
$2077 696E 646F 7720"
$6174 2074 6865 2074"
$6F70 206F 6620 7468"

```

```

$6520 6D65 6E75 2C29"
$DA22 2061 6E64 2073"
$6F20 6F6E 2E20 2054"
$6869 7320 6973 2061"
$6C6C 2064 6F6E 6520"
$6279 2074 29C4 0368"
$650D A000 99A1 009A"
$0008 000C 0000 00D8"
$0000 A000 9828 0092"
$0022 224D 4445 463B"
$2074 6865 2070 726F"
$6772 616D 2069 7473"
$656C 6620 6E65 6564"
$206E 6F74 2029 D61E"
$6265 2063 6F6E 6365"
$726E 6564 2077 6974"
$6820 6D61 696E 7461"
$696E 696E 670D A000"
$99A1 009A 0008 FFFC"
$0000 00D8 0000 A000"
$9828 00A2 0022 0A74"
$6865 206D 656E 752E"
$0DA0 0099 A100 9A00"
$08FF EC00 0000 D800"
$00A0 0098 2A10 010D"
$A000 99A1 009A 0008"
$FFDC 0000 00D8 0000"
$A000 982A 1022 596F"
$7520 6361 6E20 6164"
$6420 7468 6973 206D"
$656E 7520 746F 2079"
$6F75 7220 6F77 6E20"
$29D9 1F70 726F 6772"
$616D 7320 6279 2073"
$696D 706C 7920 636F"
$7079 696E 6720 7468"
$650D A000 99A1 009A"
$0008 FFCC 0000 00D8"
$0000 A000 9828 00D2"
$0022 224D 4445 4620"
$7265 736F 7572 6365"
$2077 6974 6820 5265"
$7365 6469 742E 2020"
$416E 2061 7229 D921"
$7469 636C 6520 6F6E"
$2074 6869 7320 7072"
$6F67 7261 6D2C 2069"
$6E63 6C75 6469 6E67"
$0DA0 0099 A100 9A00"
$08FF BC00 0000 D800"
$00A0 0098 2800 E200"
$2222 636F 6D70 6C65"
$7465 2050 6173 6361"
$6C20 736F 7572 6365"
$2063 6F64 652C 2068"
$6173 2062 29D8 1F65"
$656E 2073 7562 6D69"
$7474 6564 2074 6F20"
$4D61 6320 5475 746F"
$7220 2869 660D A000"
$99A1 009A 0008 FFAC"
$0000 00D8 0000 A000"
$9828 00F2 0022 2269"
$7420 646F 6573 6E27"
$7420 6765 7420 7075"
$626C 6973 6865 642C"
$2049 276C 6C20 7265"
$6C29 C218 6561 7365"
$2074 6865 2073 6F75"
$7263 6520 6D79 7365"
$6C66 292E A000 99A0"
$0097 A100 9600 0606"
$0000 0002 03A1 009A"
$0008 FFFA 0000 0025"

```

```

$0000 A000 9804 050D"
$0012 2800 2B00 D806"
$5769 6E64 6F77 A000"
$99A0 0097 A000 8DA0"
$0083 FF"
);
resource 'PICT' (1002,
"Attic",
purgeable) (
1125,
(7, 7, 306, 497),
$1101 A000 82A0 008C"
$0100 0A00 0700 0701"
$3201 F10A 0000 0000"
$0000 0000 0B00 1B00"
$1B44 0009 0009 0130"
$01EF 0700 0200 0248"
$A100 9600 0606 0000"
$0002 03A1 009A 0008"
$FFFA 0000 004B 0000"
$A000 9803 0003 0405"
$0D00 122B 8A2B 0E41"
$7474 6963 2053 6F66"
$7477 6172 65A0 0099"
$A000 97A1 0096 0006"
$0500 0000 0203 A100"
$9A00 0800 5C00 0000"
$E000 00A0 0098 0400"
$0D00 0C28 004D 001D"
$2241 7474 6963 2053"
$6F66 7477 6172 6520"
$6973 2061 2073 6D61"
$6C6C 204D 6163 696E"
$746F 7329 DC22 6820"
$7072 6F67 7261 6D6D"
$696E 6720 636F 6D70"
$616E 792E 2069 6E20"
$6275 7369 6E65 7373"
$29E2 010D A000 99A1"
$009A 0008 004C 0000"
$00E6 0000 A000 9828"
$005D 001D 2273 696E"
$6365 2031 3938 362E"
$2020 5765 2064 6F20"
$6120 7661 7269 6574"
$7920 6F66 2077 6F29"
$0722 7268 3B20 D249"
$6465 616C 696E 6572"
$032C 2061 2073 6861"
$7265 7761 7265 206F"
$7574 6C69 29D1 056E"
$6572 2C0D A000 99A1"
$009A 0008 003C 0000"
$00E6 0000 A000 9828"
$006D 001D 2269 7320"
$6F75 7220 6265 7374"
$2D68 6E6F 776E 2070"
$726F 6475 6374 2E20"
$2057 6520 616C 7329"
$D81B 6F20 646F 2063"
$6F6E 7472 6163 7420"
$7072 6F67 7261 6D6D"
$696E 672E 0DA0 0099"
$A100 9A00 0800 2C00"
$0000 E600 00A0 0098"
$2800 7D00 1D01 0DA0"
$0099 A100 9A00 0800"
$1C00 0000 E600 00A0"
$0098 2A10 1E57 6520"
$6361 6E20 6265 2072"
$6561 6368 6564 2062"
$7920 6D61 696C 2061"
$743A 0DA0 0099 A100"
$9A00 0800 0C00 0000"

```

```

$E600 00A0 0098 2A10"
$010D A000 99A1 009A"
$0008 FFFC 0000 00E6"
$0000 A000 982A 1022"
$2020 2020 2020 2020"
$2020 2020 2020 2020"
$2020 2020 2020 2020"
$2020 2020 2020 2041"
$7474 2991 0C69 6320"
$536F 6674 7761 7265"
$0DA0 0099 A100 9A00"
$08FF EC00 0000 E600"
$00A0 0098 2800 BD00"
$1D22 2020 2020 2020"
$2020 2020 2020 2020"
$2020 2020 2020 2020"
$2020 2020 2020 2020"
$2050 2E4F 298F 0A2E"
$2042 6F78 2032 3139"
$0DA0 0099 A100 9A00"
$08FF DC00 0000 E600"
$00A0 0098 2800 CD00"
$1D22 2020 2020 2020"
$2020 2020 2020 2020"
$2020 2020 2020 2020"
$2047 6F6C 2990 1865"
$7461 2C20 4361 6C69"
$666F 726E 6961 2020"
$2039 3331 3136 0DA0"
$0099 A100 9A00 08FF"
$CC00 0000 E600 00A0"
$0098 2800 DD00 1D01"
$0DA0 0099 A100 9A00"
$08FF BC00 0000 E600"
$00A0 0098 2A10 2257"
$6520 616C 736F 206F"
$7065 7261 7465 2061"
$2062 756C 6C65 7469"
$6E20 626F 6172 6420"
$7329 D422 7973 7465"
$6D20 6174 2028 3830"
$3529 2036 3833 2D30"
$3332 322C 2062 6574"
$7765 656E 2074 29E6"
$0368 650D A000 99A1"
$009A 0008 FFAC 0000"
$00E6 0000 A000 9828"
$00FD 001D 2268 6F75"
$7273 206F 6620 363A"
$3030 2050 4D20 616E"
$6420 323A 3030 2041"
$4D2C 2050 6163 6929"
$D822 6669 6320 7469"
$6D65 2C20 7365 7665"
$6E20 6461 7973 2061"
$2077 6565 682E 2020"
$506C 6561 29D3 0373"
$650D A000 99A1 009A"
$0008 FF9C 0000 00E6"
$0000 A000 9828 010D"
$001D 1566 6565 6C20"
$6672 6565 2074 6F20"
$6361 6C6C 2069 6E21"
$A000 99A0 0097 A100"
$9600 0606 0000 0002"
$03A1 009A 0008 FFFD"
$0000 006D 0000 A000"
$980D 000A 2B73 1722"
$A920 3139 3838 2062"
$7920 436C 6966 666F"
$7264 2053 746F 7279"
$2061 6E64 2041 7474"
$6963 29AC 0920 536F"

```

```

$6674 7761 7265 A000"
$99A0 0097 A000 8DA0"
$0083 FF"
);
/*****
Alert resource
*****/
resource 'ALRT' (1001,
    "Message", purgeable) (
    0, 0, 122, 300),
    1001,
    ( /* array: 4 elements */
    /* [1] */
    OK, visible, sound1,
    /* [2] */
    OK, visible, sound1,
    /* [3] */
    OK, visible, sound1,
    /* [4] */
    OK, visible, sound1
    )
);
/*****
Item list resource
*****/
resource 'DITL' (1001,
    "Message", purgeable) (
    /* array DITLarray: 2
    elements */
    /* [1] */
    92, 120, 112, 180),
    Button (
        enabled,
        "OK"
    ),
    /* [2] */
    (10, 10, 74, 290),
    StaticText (
        disabled,
        "Sorry! This program"
        " can open only 18 wi"
        "ndows at one time (i"
        "f it opened more, th"
        "e menu would be too "
        "long for the screen)"
        "."
    )
);
/*****
Window resources
*****/
resource 'WIND' (1001,
    purgeable) (
    0, 0, 200, 320),
    documentProc,
    invisible,
    goaway,
    0x0,
    "untitled"
);
resource 'WIND' (1002,
    purgeable) (
    0, 0, 200, 320),
    zoomDocProc,
    invisible,
    -1,
    0x0,
    "untitled"
);
/*****
Multifinder resource
*****/
resource 'SIZE' (-1) (

```

```

saveScreen,
acceptSuspendResumeEvents,
enableOptionSwitch,
cannotBackground,
MultiFinderAware,
98304,
98304
);
/*****

```

Some Standard Declarations

This first bit of code is a simple unit that I include in all my stuff these days. Every time I use a new low-memory global, I stick it in here. I keep the unit in the "PInterfaces" folder, so I can use it easily in any program.

```

(*****
Common.p
Declarations for
dynamic Window menu demo.
(c) 1988, by Clifford Story
& Attic Software
*****)
unit Common;
(*****
interface
(*****
Key codes:
*****)
const
    enterkey      = 3;
    backspace     = 8;
    tabkey        = 9;
    returnkey     = 13;
    clearkey      = 27;
    leftarrow     = 28;
    rightarrow    = 29;
    uparrow       = 30;
    downarrow     = 31;
    periodkey     = 46;

(*****
Dialog items:
*****)
    themask       = 3;
(*****
Low-memory globals:
*****)
    applscratch   = $A78;
    bootdrive     = $210;
    curappname    = $910;
    curdirstore   = $398;
    currenta5     = $904;
    findername    = $2E0;
    fsfcbilen     = $3F6;
    grayrgn       = $9EE;
    iaznotify     = $33C;
    mbarheight    = $BAA;
    menuflash     = $A24;
    resload       = $A5E;
    rom85         = $28E;
    sfsavedisk    = $214;
    sysmap        = $A58;
    windowlist    = $9D6;

(*****
Standard types:
*****)
type
    logical       = boolean;
    long          = longint;

```

```
shortpointer = ^integer;
longpointer = ^long;
```

```
(*****
end.
(*****)
```

The Menu Definition

An MDEF is called to do three things: draw the menu, hit-test the mouse location, and calculate the size of the menu rectangle.

Drawing the Menu

The menu has two parts: the first two items, which are fixed, and the list of open windows. I've hard-coded the first two items, since I know what they are. Note that if the "Zoom Front Window" item is disabled, then it is grayed out by drawing a gray rectangle over it.

To draw the list of open windows, I start with the window whose pointer is stored in the low-memory global "windowlist". The windows form a chain, with each window record including a pointer to the next one; the final window has a nil pointer in the "nextwindow" field. So I walk down this list, looking for windows to add to the menu. I'm only going to show application windows, since DA windows may not have titles, and of course I'll omit hidden windows.

Hit-Testing the Mouse

This is simple, since each menu item is 16 pixels high - just divide the distance from the top of the menu by 16 (result: a number in the range from 0 to the number of menu items minus one) and add one (since the first item is item one, not item zero).

There's a small complication: if the item is disabled, then the routine should return zero (no choice).

The "whichitem" argument is a var parameter that initially contains the value of the last choice. If the newly-calculated item is the same as the last choice, then nothing has changed, so there's nothing else to do. If it is different, then the old item should be un-highlighted and the new highlighted.

Calculating the Menu Size

This routine must compute the width and height of the menu rectangle and update the menu record accordingly. The approach here is similar to that of the draw routine: hard-code the fixed items and walk the window list for the rest, looking for the widest menu item.

Note the formulae by which the width of an item is calculated. I worked these out with screen shots and fatbits, and they (with the corresponding formula in the draw routine) will produce a menu identical to one created with the standard MDEF.

```
(*****
Window Menu.p
MDEF for dynamic Window menu demo.
(c) 1988, by Clifford Story
& Attic Software
*****)
```

```
unit WDEF;
```

```
(*****
interface
(*****)
```

```
uses mentypes, quickdraw, osintf,
    toolintf, Common;
```

```
(*****
procedure menudef(message : integer; themenu : MenuHandle;
    var menurect : Rect; hitpoint : Point; var whichitem :
integer);
(*****
implementation
(*****)
```

```
type
QDrecord = record
    randSeed : long;
    screenBits : BitMap;
    arrow : Cursor;
    dkGray : Pattern;
    ltGray : Pattern;
    gray : Pattern;
    black : Pattern;
    white : Pattern;
    thePort : GrafPtr;
end;
QDpointer = ^QDrecord;
```

```
(*****)
```

```
procedure menudraw(themenu : MenuHandle; var menurect :
Rect); forward;
procedure menuchoose(themenu : MenuHandle; var menurect :
Rect;
    hitpoint : Point; var whichitem : integer); forward;
procedure menusize(themenu : MenuHandle); forward;
```

```
(*****
($R-)
($SC+)
(*****)
```

```
procedure menudef(message : integer; themenu : MenuHandle;
    var menurect : Rect; hitpoint : Point; var whichitem :
integer);
begin
    case message of
        mDrawMsg : menudraw(themenu, menurect);
        mChooseMsg : menuchoose(themenu, menurect,
            hitpoint, whichitem);
        mSizeMsg : menusize(themenu);
    end;
end;
```

```
(*****)
```

```
function QDglobals : QDpointer;
var
    thepointer : longpointer;
begin
    thepointer := longpointer(currenta5);
    thepointer := longpointer(thepointer^);
    QDglobals := QDpointer(long(thepointer^
        - sizeof(QDrecord) + sizeof(GrafPtr)));
end;
```

```
(*****)
```

```
procedure menudraw(themenu : MenuHandle; var menurect : Rect);
var
    height : integer;
    width : integer;
    therect : Rect;
    thewindow : WindowPeek;
begin
```



```

height := menurect.top + 12;
width := menurect.left + 12;

MoveTo(width, height);
DrawString('Zoom Front Window');
MoveTo(menurect.right - CharWidth('W') - 15, height);
DrawChar(chr(17));
DrawChar('W');

if not BitTst(@themenu^.enableFlags, 30) then begin
  PenPat(QDglobals^.gray);
  PenMode(patBic);
  with menurect do
    SetRect(therect, left, top, right, top + 16);
    PaintRect(therect);
  PenNormal;
end;

height := height + 16;
MoveTo(menurect.left, menurect.top + 24);
Line(menurect.right - menurect.left, 0);

thewindow := WindowPeek(longpointer(windowlist)^);

while thewindow <> nil do
  with thewindow^ do begin
    if visible and (windowkind = userKind) then begin
      height := height + 16;
      MoveTo(width, height);
      DrawString(titlehandle^);
    end;
    thewindow := nextwindow;
  end;
end;

(*****)
procedure menuchoose(themenu : MenuHandle; var menurect :
Rect;
  hitpoint : Point; var whichitem : integer);
var
  theitem      : integer;
  therect      : Rect;
begin
  if PtInRect(hitpoint, menurect) then
    theitem := 1 + ((hitpoint.v - menurect.top) div 16)
  else
    theitem := 0;

  if not BitTst(@themenu^.enableFlags, 31 - theitem) then
    theitem := 0;

  if theitem <> whichitem then begin
    therect := menurect;
    therect.bottom := therect.top + 16 * theitem;
    therect.top := therect.bottom - 16;
    InvertRect(therect);
    if whichitem > 0 then begin
      therect.bottom := menurect.top + 16 * whichitem;
      therect.top := therect.bottom - 16;
      InvertRect(therect);
    end;
    whichitem := theitem;
  end;
end;

(*****)
procedure menusize(themenu : MenuHandle);
var
  thewidth      : integer;
  theheight     : integer;
  thewindow     : WindowPeek;
  newwidth      : integer;
begin
  TextFont(systemFont);

```

```

  TextSize(12);
  TextFace(1);

  theheight := 32;
  thewidth := StringWidth('Zoom Front Window') +
    CharWidth('W') + 39;

  thewindow := WindowPeek(longpointer(windowlist)^);

  while thewindow <> nil do
    with thewindow^ do begin
      if visible and (windowkind = userKind) then begin
        theheight := theheight + 16;
        newwidth := StringWidth(titlehandle^*) + 16;
        if newwidth > thewidth then
          thewidth := newwidth;
      end;
      thewindow := nextwindow;
    end;

    themenu^.menuHeight := theheight;
    themenu^.menuWidth := thewidth;
  end;

(*****)
end.
(*****)

```

The Main Program

This is, I think, a pretty straight-forward program, so I won't say much about it except for four features: menu maintenance, opening and closing windows, zooming windows, and window selection.

Menu Maintenance

I thought I said there wouldn't be any maintenance? Well, there is a little, but not very much.

When the length of the menu changes, I have to call CalcMenuSize to provoke the Menu Manager into calling the MDEF with an "mSizeMsg" message. If I don't do this, either there will be a blank space at the bottom of the menu, or the last item will be missing.

The size of the menu changes whenever I open or close a window, so I call CalcMenuSize at these times.

Opening and Closing Windows

The Window menu doesn't scroll; that limits it to 18 windows (20 items minus the zoom item and the dividing line). So, logically, I ought to limit the program to 18 windows as well.

If the program can open only 18 windows, there's no reason to open them dynamically and clutter up the heap with a bunch of fixed objects (WindowRecords). So instead I'll declare a global array of WindowRecords. I'll just pass the address of one of these (instead of a nil pointer) for the wStorage argument of GetNewWindow.

Ah, but now I have to keep track of which records belong to open windows, and which are free. This is actually easy to do.

First, extend the WindowRecord with a couple of new fields (see the datarecord declaration below): an integer ID and an integer "pointer" to the next record. The ID is just the index of the record in the array, and the "next" field initially points to the next record in the array (with the last one -1). Then there's

another global, an integer called FREE, which initially points to the first record in the array.

What I'm doing is keeping a list of all the free Window-Records. Initially, they're all free and, sure enough, they're all on the list. Each time I open a window, I pass the address of the Window-Record pointed to by FREE, and set FREE to that record's "next" field. That pops it off the list. When I close a window, I add the WindowRecord back to the free list, by setting its "next" field to FREE, and FREE to its ID. If FREE is ever -1, then the list is empty, and I can't open a window.

Zooming: When?

The first question is, when should I allow zooming?

Since I'm zooming with ZoomWindow (see my Corners article, April 1989, for another method that works on all ROMs), I don't want to allow zooming on the 64K ROM. So I keep a global flag, OLDROM, to tell me if the program is running on the 64K ROM, and I keep the "Zoom Front Window" item disabled if it is.

Also, I only want to zoom application windows. (I tried zooming the Alarm Clock, once, just to see what would happen...) So if the front window isn't an application window, I disable the "Zoom Front Window" item. I do this in the "anactivate" routine, disabling on deactivates and enabling on activates.

Zooming: How?

The short answer to this is: with ZoomWindow. Unfortunately, ZoomWindow ain't the smartest trap in the world, or maybe it's just to subtle for me. The problem is this "inZoomIn", "inZoomOut" nonsense. If you send it an "inZoomIn" part code, and it thinks the window is already zoomed in, it will do nothing. So I have to figure out when it thinks it's zoomed in, and when it thinks it's zoomed out. Maybe there's an easy way; for my part, this is the last time I plan to use ZoomWindow.

This wouldn't trouble me if I was only zooming from the zoom box, since FindWindow will tell me which part code to use. The problem arises because I'm also zooming from the menu.

What I decided to do is keep a flag in the reference constant of each window, setting it when the window is zoomed out, and clearing it otherwise. So when the time comes to zoom, I just look at the flag and zoom in if it's set, and out if it's cleared. This is done in the routine "zoomthewindow".

Now I've got a flag to maintain. Besides the obvious in the "zoomthewindow" routine, I also must initialize the flag (in "donew"), and clear it if the window is dragged (in "clickindrag") or re-sized from the grow box (in "clickingrow").

Selecting a Window from the Menu

There is a little bit of work to do when a selection is made from the window list in the Window menu. After all, what the Menu Manager returns is an integer, not a WindowPtr. I have to walk the window list, just like in the MDEF, until I find the correct window, and then select it.

```
(*****
Window.p
Demo of a dynamic Window menu.
(c) 1988, by Clifford Story
& Attic Software
*****)
```

```
program Window;
```

```
(*****
uses mentypes, quickdraw, osintf,
    toolintf, packintf, Common;
```

```
(*****
Program constants:
*****)
```

```
const
```

```
    applenum    = 1001;
    aboutitem    = 1;
    atticitem    = 2;
```

```
    filenum     = 1002;
    newitem      = 1;
    closeitem    = 2;
    quititem     = 4;
```

```
    editnum     = 1003;
    undoitem     = 1;
    cutitem      = 3;
    copyitem     = 4;
    pasteitem    = 5;
    clear        = 6;
```

```
    windnum     = 1004;
    zoomitem     = 1;
```

```
    messagedialog = 1001;
```

```
    windownum    = 1001;
    zoomwindnum   = 1002;
```

```
    hoffset      = 32;
    voffset       = 20;
```

```
(*****
Program types:
*****)
type
```

```
    datarecord   = record
        dummy    : WindowRecord;
        recordid  : integer;
        next      : integer;
    end;
    datapointer  = ^datarecord;
```

```
(*****
Program variables:
*****)
```

```
var
```

```
    APPLEMENU    : MenuHandle;
    FILEMENU     : MenuHandle;
    EDITMENU      : MenuHandle;
    WINDOWMENU    : MenuHandle;
```

```
    MENUHEIGHT   : integer;
    DRAGRECT      : Rect;
    GROWRECT      : Rect;
    SCREENRECT    : Rect;
```

```
    WINDOWS      : array [0..17] of datarecord;
```

```

FREE      : integer;

OLDROM    : logical;
COLUMNS  : integer;
ROWS      : integer;
WINDOWCOUNT : integer;

DONE      : logical;
JEVENT    : logical;
MAINEVENT : EventRecord;
(*****
procedure _datainit; external;
(*****
($R-)
($SC+)
(*****
procedure panic;
begin
    ExitToShell;
end;

(*****
procedure centerdialog( thetype : OSType; theid : integer);
var
    thehandle : AlertTHndl;
begin
    thehandle := AlertTHndl(GetResource(thetype, theid));
    HLock(Handle(thehandle));
    with thehandle do begin
        with boundsRect do
            SetRect(boundsRect, 0, 0, right - left, bottom - top);
        with screenBits.bounds,
            boundsRect.botrigh do
                OffsetRect(boundsRect, (right - left - h) div 2,
                    (bottom - top - v + 2 * MENUHEIGHT) div 3);
    end;
    HUnlock(Handle(thehandle));
end;

(*****
procedure initmac;
begin
    MaxApplZone;
    InitGraf(@thePort);
    InitFonts;
    InitWindows;
    InitCursor;
    InitMenus;
    TEInit;
    InitDialogs(@panic);
    UnloadSeg(@_datainit);
end;

(*****
procedure setupmenus;
begin
    APPLEMENU := GetMenu(applenum);
    AddResMenu(APPLEMENU, 'DRVr');
    InsertMenu(APPLEMENU, 0);
    FILEMENU := GetMenu(filenum);
    InsertMenu(FILEMENU, 0);
    EDITMENU := GetMenu(editnum);
    InsertMenu(EDITMENU, 0);
    WINDOWMENU := GetMenu(windownum);
    InsertMenu(WINDOWMENU, 0);
    DrawMenuBar;
end;

(*****
procedure initglobals;
var
    index : integer;
begin
    for index := 1 to 10 do

```

```

    MoreMasters;

    OLDROM := BitTst(Ptr(rom85), 0);

    if OLDROM then
        MENUHEIGHT := 20
    else
        MENUHEIGHT := shortpointer(mbarheight)*;

    if GetResource('PACK', 0) = nil then
        JEVENT := false
    else
        JEVENT := (NGetTrapAddress($A860, ToolTrap)
            <> NGetTrapAddress($A89F, ToolTrap));

    with screenBits.bounds do begin
        SetRect(DRAGRECT, left + 5, top + MENUHEIGHT + 5,
            right - 5, bottom - 25);
        SetRect(GROWRECT, 160, 100, right - left - 10, bottom
            - top - MENUHEIGHT - 10);
        SetRect(SCREENRECT, left + 5, top + MENUHEIGHT + 25,
            right - 5, bottom - 5);
        COLUMNS := 1 + ((right - left - 330) div hoffset);
        ROWS := 1 + ((bottom - top - MENUHEIGHT - 230) div voffset);
    end;
    for index := 0 to 17 do
        with WINDOWS[index] do begin
            recordid := index;
            next := index + 1;
        end;
    WINDOWS[17].next := -1;

    FREE := 0;
    WINDOWCOUNT := 0;
    DONE := false;
end;

(*****
procedure clickapplenue(theitem : integer);
var
    itemname : Str255;
    savedport : GrafPtr;
    dummy : integer;
    newport : GrafPort;
    thepicture : PicHandle;
    therect : Rect;
begin
    if theitem > 3 then begin
        GetItem(APPLEMENU, theitem, itemname);
        GetPort(savedport);
        dummy := OpenDeskAcc(itemname);
        SetPort(savedport);
    end else if theitem < 3 then begin
        InitCursor;
        GetPort(savedport);
        OpenPort(@newport);
        SetPort(@newport);
        thepicture := PicHandle(GetResource('PICT', 1000 + theitem));
        with thepicture do
            with thepicture do
                SetRect(therect, 0, 0, right - left, bottom - top);
            with screenBits.bounds, therect.botrigh do
                OffsetRect(therect, (right - left - h) div 2,
                    (bottom - top - v) div 3);
            DrawPicture(thepicture, therect);
            repeat until Button;
            ClosePort(@newport);
            EnableItem(EDITMENU, 0);
            DrawMenuBar;
            PaintBehind(WindowPeek(FrontWindow), RgnHandle(
                longpointer(grayrgn)));
            SetPort(savedport);
            FlushEvents(everyEvent, 0);
        end;
    end;

```

```

(*****)
procedure placewindow(thewindow : WindowPtr);
var
    left      : integer;
    top       : integer;
begin
    left := 5 + hoffset * (WINDOWCOUNT mod COLUMNS);
    top := 5 + MENUHEIGHT + voffset * (1 + (WINDOWCOUNT mod ROWS));
    MoveWindow(thewindow, left, top, true);
end;

(*****)
procedure donew;
label
    100;
var
    dummy      : integer;
    windowtype : integer;
    thewindow   : WindowPtr;
    thestring   : Str255;
begin
    if FREE < 0 then begin
        InitCursor;
        centerdialog('ALRT', messagedialog);
        dummy := Alert(messagedialog, nil);
        goto 100;
    end;

    if OLDROM then
        windowtype := windownum
    else
        windowtype := zoomwindownum;

    thewindow := GetNewWindow(windowtype,
        @WINDOWS[FREE], WindowPtr(-1));
    placewindow(thewindow);

    WINDOWCOUNT := WINDOWCOUNT + 1;
    NumToString(WINDOWCOUNT, thestring);
    SetWindowTitle(thewindow, concat(
        'Window Without a Title #', thestring));

    ShowWindow(thewindow);

    CalcMenuSize(WINDOWMENU);

    FREE := WINDOWS[FREE].next;
100: end;

(*****)
procedure doclose(thepeek : WindowPeek);
begin
    if thepeek.windowkind < 0 then
        CloseDeskAcc(thepeek.windowkind)
    else begin
        CloseWindow(WindowPtr(thepeek));
        with datapointer(thepeek) do begin
            next := FREE;
            FREE := recordid;
        end;
        CalcMenuSize(WINDOWMENU);
    end;
end;

(*****)
procedure doquit;
var
    thelong      : longpointer;
    thepeek      : WindowPeek;
begin
    thelong := longpointer(windowlist);
    thepeek := WindowPeek(thelong);

```

```

    while thepeek <> nil do begin
        doclose(thepeek);
        thepeek := thepeek.nextwindow;
    end;
    DONE := true;
end;

(*****)
procedure clickfilemenu(theitem : integer);
begin
    case theitem of
        newitem      : donew;
        closeitem    : doclose(WindowPeek(FrontWindow));
        quititem     : doquit;
    end;
end;

(*****)
procedure zoomthefront(thefront : WindowPtr);
begin
    if GetWRefCon(thefront) = 0
        then begin
            ZoomWindow(thefront, inZoomOut, true);
            SetWRefCon(thefront, 1);
        end else begin
            ZoomWindow(thefront, inZoomIn, true);
            SetWRefCon(thefront, 0);
        end;
end;

(*****)
procedure clickwindowmenu(theitem : integer);
var
    thewindow      : WindowPeek;
begin
    if theitem = 1 then
        zoomthefront(FrontWindow)
    else if theitem > 2 then begin
        theitem := theitem - 2;
        thewindow := WindowPeek(longpointer(windowlist));
        while (not thewindow.visible)
            or (thewindow.windowkind <> userKind) do
            thewindow := thewindow.nextwindow;
        while theitem > 1 do begin
            theitem := theitem - 1;
            repeat
                thewindow := thewindow.nextwindow;
            until thewindow.visible and (thewindow.windowkind =
                userKind);
        end;
        SelectWindow(WindowPtr(thewindow));
    end;
end;

(*****)
procedure checkmenu(thefront : WindowPeek);
begin
    DisableItem(EDITMENU, 0);
    DisableItem(WINDOWMENU, zoomitem);
    if thefront = nil then
        DisableItem(FILEMENU, closeitem)
    else begin
        if thefront.windowkind <> userKind then
            EnableItem(EDITMENU, 0)
        else if not OLDROM then
            EnableItem(WINDOWMENU, zoomitem);
            EnableItem(FILEMENU, closeitem);
        end;
    end;
end;

(*****)
procedure clickinmenu;
var
    choice      : long;

```

```

begin
  checkmenu(WindowPeek(FrontWindow));
  choice := MenuSelect(MAINEVENT.where);
  case HiWord(choice) of
    applenum : clickapplenum(LoWord(choice));
    filenum  : clickfilemenu(LoWord(choice));
    editnum  : if SystemEdit(LoWord(choice) - 1) then;
    windnum  : clickwindowmenu(LoWord(choice));
  end;
  HiliteMenu(0);
end;

(*****)
procedure clickindrag(thewindow : WindowPtr);
begin
  DragWindow(thewindow,
    MAINEVENT.where, DRAGRECT);
  SetWRefCon(thewindow, 0);
end;

(*****)
procedure clickingrow(thewindow : WindowPtr);
var
  newsize : long;
begin
  newsize := GrowWindow(thewindow, MAINEVENT.where, GROWRECT);
  if newsize <> 0 then begin
    InvalRect(thewindow^.portRect);
    SizeWindow(thewindow, LoWord(newsize), HiWord(newsize), true);
    SetWRefCon(thewindow, 0);
  end;
end;

(*****)
procedure clickingaway(thewindow : WindowPtr);
begin
  if TrackGoAway(thewindow, MAINEVENT.where) then
    doclose(WindowPeek(thewindow));
end;

(*****)
procedure clickinzoom(thewindow : WindowPtr; thezoom :
integer);
begin
  if TrackBox(thewindow, MAINEVENT.where, thezoom) then
    zoomthefwindow(thewindow);
end;

(*****)
procedure aclick;
var
  location : integer;
  thewindow : WindowPtr;
begin
  location := FindWindow(MAINEVENT.where, thewindow);
  case location of
    inDesk : SysBeep(1);
    inMenuBar : clickinmenu;
    inSysWindow : SystemClick(MAINEVENT, thewindow);
    inContent : SelectWindow(thewindow);
    inDrag : clickindrag(thewindow);
    inGrow : clickingrow(thewindow);
    inGoAway : clickingaway(thewindow);
    inZoomIn : clickinzoom(thewindow, inZoomIn);
    inZoomOut : clickinzoom(thewindow, inZoomOut);
  end;
end;

(*****)
procedure akey;
var
  charcode : integer;
  choice : long;
begin

```

```

  if BitAnd(MAINEVENT.modifiers, cmdKey) <> 0 then begin
    charcode := BitAnd(MAINEVENT.message, charCodeMask);
    checkmenu(WindowPeek(FrontWindow));
    choice := MenuKey(chr(charcode));
    if choice <> 0 then begin
      case HiWord(choice) of
        applenum : clickapplenum(LoWord(choice));
        filenum  : clickfilemenu(LoWord(choice));
        editnum  : if SystemEdit(LoWord(choice) - 1) then;
        windnum  : clickwindowmenu(LoWord(choice));
      end;
      HiliteMenu(0);
    end;
  end;

(*****)
procedure anactivate(thewindow : WindowPtr);
var
  savedport : GrafPtr;
begin
  SetPort(thewindow);
  DrawGrowIcon(thewindow);
end;

(*****)
procedure anupdate(thewindow : WindowPtr);
var
  savedport : GrafPtr;
begin
  GetPort(savedport);
  SetPort(thewindow);
  BeginUpdate(thewindow);
  ClipRect(thewindow^.portRect);
  EraseRect(thewindow^.portRect);
  DrawGrowIcon(thewindow);
  EndUpdate(thewindow);
  SetPort(savedport);
end;

(*****)
procedure mainloop;
var
  dummy : logical;
begin
  repeat
    if JEVENT then
      dummy := waitnextevent(everyEvent, MAINEVENT,
        GetCaretTime, nil)
    else begin
      SystemTask;
      dummy := GetNextEvent(everyEvent, MAINEVENT);
    end;
    if dummy then begin
      case MAINEVENT.what of
        mouseDown : aclick;
        keyDown : akey;
        activateEvt :
          anactivate(WindowPtr(MAINEVENT.message));
        updateEvt :
          anupdate(WindowPtr(MAINEVENT.message));
      end;
    end;
  until DONE;
end;

(*****)
begin
  initmac;
  setupmenus;
  initglobals;
  mainloop;
end.

```



Mac OOPs

Introduction to MacApp



David E. Smith
Editor & Publisher
MacTutor
Volume 5, Number 8

Getting Started in MacApp

Apple Computer has now released MPW 3.0 final and MacApp 2.0b9, making Apple the first company to release a complete object-oriented design platform for its family of Macintosh workstations. After several years of work, the early development efforts that started with Smalltalk, Lisa Clascal, and early versions of MacApp have now paid off in a very stable and robust interpretation of the Macintosh toolbox. MacApp re-defines much of the standard toolbox and the familiar toolbox objects such as windows, events, and menus, as objects, able to respond to commands. Further refinements of the concept of windows and menus in particular extend these toolbox structures into the world of objects in a more flexible and powerful way than before. In this article, we will introduce MacTutor readers to object-oriented programming and MacApp in our new Mac OOPs column, a regular feature each month in the new MacTutor.

MPW 3.0 Final

To run MacApp 2.0b9, you must have the final release of MPW 3.0. Without it, the compiler cannot compile some of the MacApp source code, and some of the needed new Pascal interfaces are missing in prior beta versions. The final release of MacApp 2.0 is currently waiting delivery of new manuals but otherwise is supposed to be identical to 2.0b9, although reports this spring indicated that there were several "b9" versions running around at the last minute.

MacApp is delivered in source code format and is compiled in either "debug" or "nodebug" versions when you compile your own source code by specifying the MPW "-autobuild" option. A paper copy of the source code is also available. Contact APDA, now a part of Apple Computer, for details on how to order MPW 3.0 final and MacApp 2.0. This version really represents what MacTutor considers the first stable and universally usable version of MacApp, and we highly recommend that you get on the MacApp bandwagon with this version, if you haven't already. Currently MacApp only runs with MPW Object Pascal, but header files are being developed to interface the MacApp object libraries with Apple's new C++ Compiler, and Think Technologies is preparing a new version of Lightspeed Pascal to run with MacApp.

Objects & C

C++ has been delayed by AT&T, which is controlling its release. Apple has been allowed only 200 copies until the final release from AT&T, which just took place at the end of June. It will be several months from that date before Apple can prepare it for release to developers. C++ was chosen over the other

object-oriented C pre-processors because Apple believes it will be a defacto standard in the industry. Think Technologies is also preparing another release of their Lightspeed C compiler, which will include a form of Objective C, a slightly different object-oriented version of C. Most C people feel Objective C has a much more natural syntax than C++, which places power and flexibility before everything else, at the expense of a straight forward syntax. The C world in general really does not have a good syntax for objects in comparison to Object Pascal, which has only one standard implementation in contrast to the three or four implementations floating around in the C world.

What is MacApp?

MacApp is a set of object classes that re-defines the Macintosh user interface in the form of objects. The keyword OBJECT in Object Pascal implements a new kind of RECORD data structure that bundles together variables (i.e. fields), with procedures (i.e. methods), which act on those variables. Together, the fields and methods form an object and by virtue of those bundled methods, the object can "respond" to commands. Objects can inherit fields and methods from other objects, and they can have methods which override the methods of the objects they are descended from. MacApp defines TObject(Object) as the basis for all the objects defined in MacApp. TObject differs from its inherited Object class by having a method to clone itself. From TObject all the other MacApp objects are developed. In MacApp, it is customary to start all object names with a "T" and all data fields within objects with "f", to distinguish object variables from methods. Figure 1 shows the object hierarchy of MacApp. The indentation shows the nested hierarchy of the classes.

A Simple MacApp Program

In our demo program this month, we present a simple MacApp program that calls SysEnvirons and displays all the information about what type of Mac we are running on, and calls PrGeneral to display the resolution of the currently selected printer driver. These two functions are shown in the Sample Menu as "Query Mac" and "Query Printer". Our program is based on a similar program written in Lightspeed Pascal and published by myself and Dave Kelly a few months ago in MacTutor. That program took several weeks of hard work to write. This MacApp version took a few hours and is much more functional than the original. The speed with which programs can be implemented that follow the expected Macintosh User Interface Guidelines is the major advantage for using MacApp. This is because the objects you use in your program inherit from MacApp object classes, methods that know how to respond to

events, menus and errors. In the case of this month's program, I wrote a draw method that calls SysEnviorns and PrGeneral and displays the results. Everything else was standard MacApp code that gave the program a long list of features: multi-window, Multifinder compatible, desk accessory support, clipboard support, Open, Save, Print, scrolling, grow window, zoom window and an about box. Writing all that functionality easily takes a few weeks or more, but with MacApp it all comes for free by virtue of the methods objects inherit from the pre-defined MacApp object classes.

MacApp Classes

As figure 1 shows, from TObjct, MacApp defines an abstract object called TEvtHandler and from this object, the familiar Macintosh objects are descended, including the application object, the document object, the window object and a new thing called the view object. In MacApp, the application program itself is considered an object that "knows" how to open documents. Documents in turn are objects which "know" how to read and write themselves to disk. Views are objects that "know" how to display some portion of the document on the screen, in a window. Note that a window itself, is defined as a view object even though we think of windows as semi-hardware things that display views of our data. Think of a view as a further refinement of the window, allowing a more flexible and general way to present data in the window and try not to think of the window as a view, even though it is, because that's confusing.

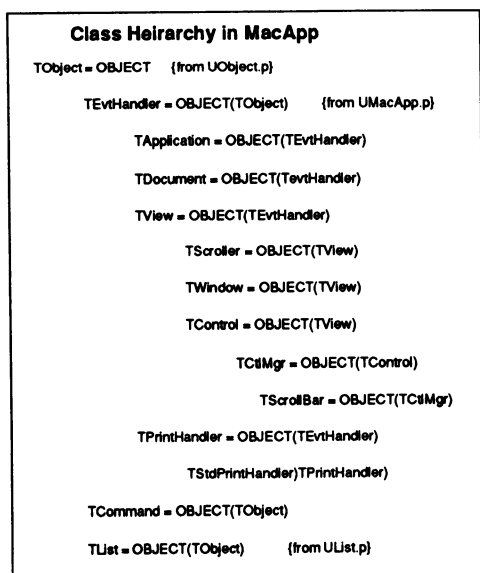


Figure 1

Logical Hierarchy

Figure 2 attempts to show the logical or observable hierarchy of the various MacApp classes. At the top are the application object and the document object. Then come the various objects which allow information to be displayed on the screen. Logically, the window object comes first. Within the window, we define a scroller object and then a view object. The scroller object

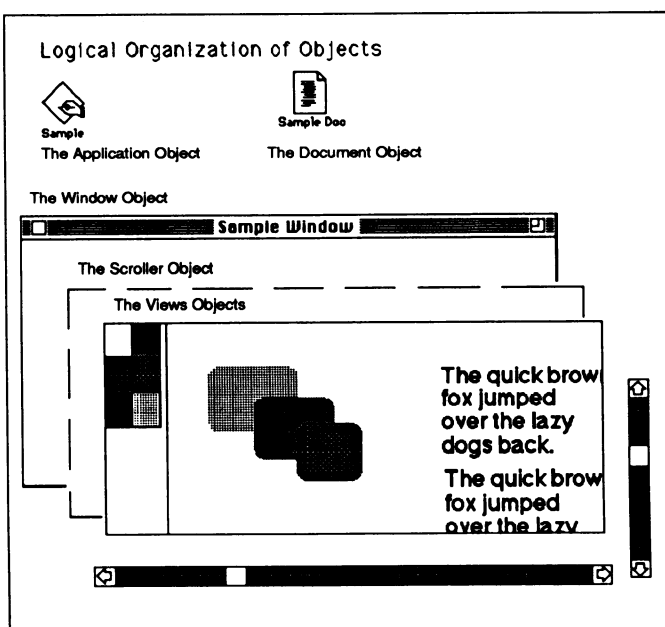


Figure 2

allows us to define regions in the window that will be scrollable. The view object sits on top of the scroller object and displays the information in the document. Thus the view and hence the document is scrollable by virtue of the scroller object. Scroll bars attached to the window interact with the scroller object to control the scrolling of the view. As soon as you define a view and draw into it, your application becomes multi-window and scrollable by virtue of the view's and window's inherited behavior. By implementing views and scroller objects, you can define multiple independent regions in your window that can scroll and display information from the document object. Printing comes for free also, by simply having a view draw itself independent of a window object.

Event Handling

In a normal Macintosh program, you init the managers and enter an event loop, waiting for a mouse click or a menu command. Your program is built-up in top-down structured fashion around the event loop with procedures that implement each event function and each menu item. The code follows the event tree structure rather than the data generated by the program.

In MacApp, the event loop is hidden, as a part of the TEvtHandler object's methods that our objects inherit from MacApp. So where does our code go? We break it up and stick part of the functionality of our program with each object that is logical for that object to understand. Figure 3 shows how this works. Our TApplication object, the application itself, has methods that implement functions the program should do if no document has been opened, such as putting up an about box or opening a document. The document object has methods that save and print itself. The window object has methods that zoom and drag the window. And a view object that implements a palette of tools has a method that selects a tool from the palette.

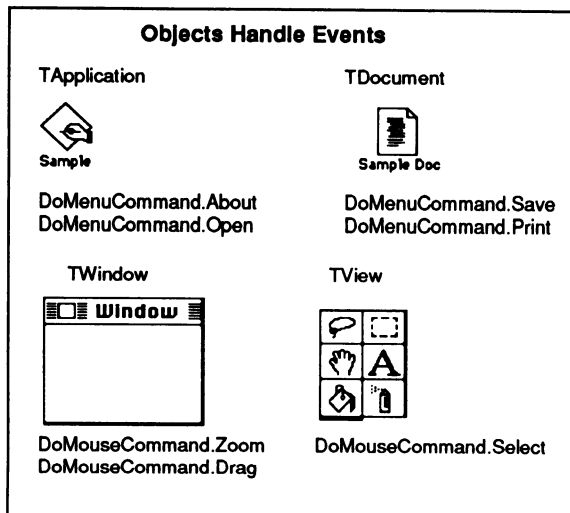


Figure 3

Menu Commands

A major change in MacApp is that menus are linearized. Instead of a menu and menu items, MacApp returns a single “menu command” number. Objects respond to menu commands by having a “DoMenuCommand” method that can process the various menu commands that object should understand. Commands are generalized by this command number concept so that commands themselves are objects created by your program. This allows a command to respond to another command to “Undo” itself or “Redo” itself. Menu command objects are created by “DoMenuCommand” methods. Mouse command objects are created by “DoMouseCommand” methods. Typing command objects are created by “DoKeyCommand” methods. Each of your objects has DoMenu, DoMouse or DoKey command methods appropriate for that object, as figure 3 illustrates.

Our Sample Program

Our program defines three objects; an application object, a document object and a view object. Our view object has a draw method that draws the results of the calls to SysEnviron and PrGeneral. The inherited behavior of these three objects implement all the other Macintosh User Interface features.

The custom in MacApp is to have a main program, shown at the start of the program listing, begin with “M”. Then the unit for the program, which begins with “U” follows. The USample unit lists in the interface part, all of the object definitions and as is shown in the listing, there are three objects defined here. Following the object declarations, the object methods are listed in the implementation part, which is broken out as a separate include file called USample.inc1. It would be more logical I would think to define a single object in each unit and to have both the object’s declaration and it’s methods in that unit. But the USES statement in Pascal might prevent this, since UNITS cannot use each other, which might be required if each unit contained the declaration and methods of a single object. Breaking up the methods and declaration of the object into two separate files seems to destroy the whole point of having objects. I would welcome some

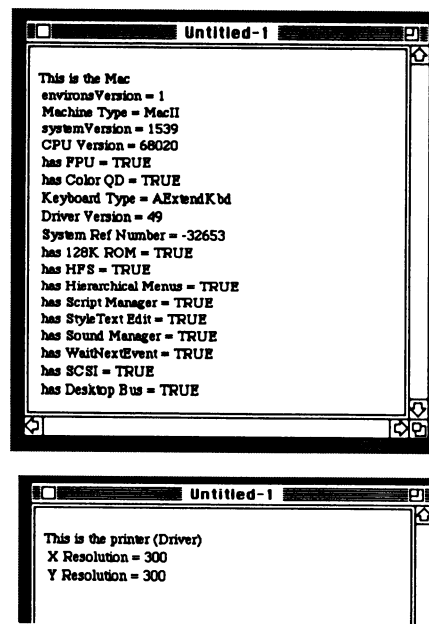


Figure 4. Mac and Printer Queries

comments from our readers on this point.

The Main Program

All main programs in MacApp look the same and look very much like this one. You simply init the Macintosh toolbox stuff, and the MacApp object stuff and then create an application object. Then you call the application object’s init method and invoke the application object’s run method. The run method is inherited from the fact that an application object is a TEvtHandler class object. If the MacApp error routine, FailNil, detects a problem in creating the application object (like not enough memory), then another MacApp routine, StdAlert, puts up an appropriate error dialog and gracefully exits. Thus our program gets nice error checking for free and doesn’t blow up on the user if he is running on a machine that cannot be supported. This is one of the main features of using MacApp— the built-in error checking and pre-defined dialog boxes for dealing with those errors. The call to ValidateConfiguration is a MacApp version of the SysEnviron trap and determines if we can run or not. By following this sample main program, you can create robust programs that will run on any Macintosh, at least long enough to put up a dialog and tell the user why they can’t!

USampe.p

Our one and only unit in this program is made up of two files: the interface part in USample.p and the implementation part in USample.inc1.p. When we declare our three objects of TApplication, TDocument and TView, we declare the data fields if any the object has, and the methods of that object. The application object has only two methods: ISampleApplication and DoMakeDocument. This allows the application object to initialize itself and make a new document when the user selects New or Open from the File menu.

The document object has several methods which override MacApp default methods for it's inherited TDocument object class. Again, it has a method to initialize itself (ISampleDocument), create a view (DoMakeViews), create a window (DoMakeWindows), setup the menus (DoSetupMenus) and execute menu commands (DoMenuCommands). All of the code for these methods are simply taken from the MacApp supplied demo program. You don't have to write them. You simply modify them for your own program. The only two methods I wrote for our sample are SetCmd and GetCmd, which set and return the current menu command, either "Query Mac" or "Query Printer". The custom in object oriented programming is that objects should not directly read or modify another object's data fields, so you write little methods to set and return the value of those fields. "fDataCmd" is the field in TSampleDocument that contains the current menu command. The other field in our document object is the document's view: fSampleView.

The view object has two methods: ISampleView to initialize the view, and Draw, where the actual drawing for the application takes place.

USample.inc1.p

Once the objects are defined, we then write the code for the methods of each object. That listing is shown in USample.inc1.p, an include file for USample.p. Here all the methods and other Pascal procedures are listed. MacApp provides a number of alternatives to toolbox trap calls. One of these is NewSimpleWindow. In our document's DoMakeWindows method, we call NewSimpleWindow, which creates a Macintosh window and returns a handle to a window object. By using the SELF keyword, we tell NewSimpleWindow to make the connection between us (our document) and our view (fSampleView). The document has a view and the view is connected to a window manager window by this call. The other MacApp goodie is a routine that staggers the windows as they are created called SimpleStagger. This is a method that window objects inherit.

DoSetupMenus

In MacApp, menu handling is particularly easy. Remember how much trouble it is to check and uncheck the currently selected menu item? In our document's DoSetupMenus method, this is very easy. We call a MacApp routine called Enable to enable the menu items. MacApp constantly disables everything, so each object has to enable the items appropriate for that object. Then we call something I wrote— our document's GetCmd method to return the currently selected menu item. Then a call to another MacApp routine called EnableCheck simply checks the menu item if it equals the result returned from GetCmd and unchecks it otherwise. Four lines of code and you have a consistent checking and unchecking of a menu!

Inheritance

Remember that I said a lot of the functionality of the program for opening files and printing and saving was already done for me? Here is how it works. When my document's DoMenuCommand method is called, I check my Sample Menu items, Query

Mac and Query Printer, against the command number passed to me by the MacApp method for handling events. If the user has selected the Query Mac item, I set the current command field in my document to indicate Mac and tell by document's view to go re-draw itself. Sounds nasty doesn't it? If the Query Printer item is chosen, I set the command to indicate Printer and again tell my document's view to re-draw itself. When the Draw method of the view is called, the documents command field will indicate whether to draw the Mac info or draw the Printer info. That is all I do. All the other functionality is inherited from objects from which my document is descended from. So I simply call INHERITED DoMenuCommand to pass the buck back up the chain of objects and all the other menu items I never wrote will be executed by code I never wrote! That's the freebie stuff from MacApp!

TSampleView.Draw

The one thing I did write for this program was a draw method for my one and only view. When an update event takes place, each view's draw method is called. So you get free updating of your window's, correctly done, I might add, my simply providing each view with a Draw method. We check the document's command field by calling the documents GetCmd method, which returns the data field in our document that is storing the current menu selection. If it's Mac, we draw the Mac and if it's Printer, we draw the printer.

To draw the Mac, we call SysEnvirons and DrawString to display the results of all the Mac features we have. This code is very similar to that which we published a few months ago on how to call SysEnvirons. In System 7, this routine is being replaced by yet another attempt to describe the environment. Instead of SysEnvirons, we will have Gestalt!

PrGeneral

To draw the printer, we call PrGeneral and display the horizontal and vertical resolution of the printer. At first, I thought PrGeneral would be really neat. But now I think it is useless. All it does is query the currently selected Chooser print driver. If you are using the normal Apple LaserWriter driver to drive your high resolution imagesetter, PrGeneral still returns 300 by 300 so it is not really telling you the actual resolution of the printer at all. To do that, you have to write an Appletalk routine to talk to the Appletalk device directly. I didn't have time to do this for this article, but in a future column perhaps we can look at how you actually talk to the real thing instead of just the driver.

Resources

The final file in our program listing is the resource file. You'll note that MacApp resource files have a few more strange resources in them. These are the MacApp menu resources that are created to linearize the menu and menu item number into a single menu command number. MacApp creates these automatically by "Post-Rezing" the resource file during the build procedure. If you look at your application with ResEdit, you'll find a whole bunch of dialogs you didn't write either. These are the error checking stuff that MacApp provides for you free of charge.

Watch for more on how to program in MacApp in our next column on Mac OOPs and send in your comments on this new column so we can provide you with the kinds of things you are interested in.

File MSample.p:

PROGRAM Sample;

```
($MC68020-) (The main program must be universal code)
($MC68881-)
```

USES

```
UMacApp, UPrinting, (MacApp stuff)
USample; (Our stuff)
```

VAR

```
gSampleApplication: TSampleApplication;
($S Main)
```

BEGIN

```
InitToolBox; (essential toolbox and utilities)
IF ValidateConfiguration(gConfiguration) THEN
  BEGIN (make sure we can run)
    InitUMacApp(8); ( 8 calls to MoreMasters)
    InitUPrinting; (init MacApp units)

    New(gSampleApplication); (create an app object)
    FailNil(gSampleApplication); (enough memory)
    gSampleApplication.ISampleApplication;
    gSampleApplication.Run; (run application)
  END
ELSE
  StdAlert(phUnsupportedConfiguration);
END.
```

File USample.p:

UNIT USample;

INTERFACE

USES

```
UMacApp, UPrinting, UMAUtil, (MacApp Stuff)
ToolUtils, Resources, Fonts, Packages, Printing;
```

TYPE

```
TSampleApplication = OBJECT(TApplication)
  PROCEDURE TSampleApplication.ISampleApplication;
  FUNCTION TSampleApplication.DoMakeDocument(
    itsCmdNumber: CmdNumber): TDocument; OVERRIDE;
END;
```

```
TSampleDocument = OBJECT(TDocument)
```

```
  fDeviceCmd: Integer;
  fSampleView: TSampleView;
  PROCEDURE TSampleDocument.ISampleDocument;
  PROCEDURE TSampleDocument.DoMakeViews(forPrinting:
    BOOLEAN); OVERRIDE;
  PROCEDURE TSampleDocument.DoMakeWindows; OVERRIDE;
  PROCEDURE TSampleDocument.DoSetupMenus; OVERRIDE;
  FUNCTION TSampleDocument.DoMenuCommand(aCmdNumber:
    CmdNumber): TCommand; OVERRIDE;
```

```
  PROCEDURE TSampleDocument.SetCmd(CmdID: INTEGER);
  FUNCTION TSampleDocument.GetCmd: INTEGER;
END;
```

```
TSampleView = OBJECT(TView)
```

```
  fSampleDocument: TSampleDocument;
  PROCEDURE TSampleView.ISampleView(itsDocument: TSample-
    Document);
  PROCEDURE TSampleView.Draw(area: Rect); OVERRIDE;
```

END;

IMPLEMENTATION

```
($I USample.incl.p)
END.
```

File USample.incl.p:

CONST (private)

```
kFileType = '????'; (File Type)
kSignature = 'Samx'; (Creator name)
kSquareDots = TRUE; (for ImageWriter printer)
kFixedSize = TRUE;
kWindRsrcID = 1001;
```

```
kPrinterStringID = 1001; (string resource ID)
kMacStringID = 1002; (string resource ID)
kFontStringID = 1003;
```

```
cMac = 2001; (menu command number)
cPrinter = 2002; (menu command number)
```

VAR (private)

```
hPos, vPos, Leading: Integer; (Used in Draw proc)
```

```
(=====)
```

```
($S Ainit)
```

```
PROCEDURE TSampleApplication.ISampleApplication;
```

BEGIN

```
  SELF.IApplication(kFileType);
```

END;

```
(-----)
```

```
($S AOpen)
```

```
FUNCTION TSampleApplication.DoMakeDocument(
  itsCmdNumber: CmdNumber): TDocument;
```

OVERRIDE;

VAR

```
aSampleDocument: TSampleDocument;
```

BEGIN

```
  NEW(aSampleDocument); (create document object)
  FailNil(aSampleDocument); (enough memory)
  aSampleDocument.ISampleDocument; (init doc object)
  DoMakeDocument := aSampleDocument; (result is doc)
```

END;

```
(=====)
```

```
($S AOpen)
```

```
PROCEDURE TSampleDocument.ISampleDocument;
```

BEGIN

```
  SELF.IDocument(kFileType, kSignature, kUsesDataFork,
    NOT kUsesRsrcFork, NOT kDataOpen, NOT kRsrcOpen);
  SELF.fSavePrintInfo := TRUE; (print record saved)
  SELF.fDeviceCmd := cMac;
```

END;

```
(-----)
```

```
($S AOpen)
```

```
PROCEDURE TSampleDocument.DoMakeViews(forPrinting: BOOLEAN);
OVERRIDE;
```

VAR

```
aSampleView: TSampleView;
aStdHandler: TStdPrintHandler;
```

BEGIN

```
  NEW(aSampleView); (create a view object)
  FailNil(aSampleView); (enough memory)
  aSampleView.ISampleView(SELF);
  (pass doc that created this view)
  SELF.fSampleView := aSampleView;
  (most documents keep a reference to their views)
```

```

New(aStdHandler);      (create a print handler)
FailNIL(aStdHandler);  (enough memory)
aStdHandler.IStdPrintHandler(SELF, aSampleView, NOT kSquare-
Dots, kFixedSize, kFixedSize);
END;

```

```

(-----)
($S AOpen)
PROCEDURE TSampleDocument.DoMakeWindows; OVERRIDE;
VAR
  aWindow: TWindow;
BEGIN
  aWindow := NewSimpleWindow(kWindRsrcID, kWantHScrollBar,
kWantVScrollBar, SELF, (doc) SELF.fSampleView); (view)
  aWindow.SimpleStagger(kStdStaggerAmount, kStdStaggerAmount,
gStdStaggerCount);
END;

```

```

(-----)
($S ARes)
PROCEDURE TSampleDocument.DoSetupMenus; OVERRIDE;
VAR
  SampleIndex: Integer;
  SampleCmd: Integer;
BEGIN
  INHERITED DoSetupMenus; (always do this)
  Enable(cMac, TRUE);
  Enable(cPrinter, TRUE); (enable this menu item)
  SampleCmd:=SELF.GetCmd;
  FOR SampleIndex:=cMac TO cPrinter DO
    EnableCheck(SampleIndex, TRUE, (SampleIndex = SampleCmd));
END;

```

```

(-----)
($S ARes)
FUNCTION TSampleDocument.DoMenuCommand(aCmdNumber:
CmdNumber):TCommand;OVERRIDE;
BEGIN
  DoMenuCommand := gNoChanges; (do this if not Undoable)
  CASE aCmdNumber OF
    cMac:
      BEGIN
        SELF.SetCmd(cMac);
        SELF.fSampleView.ForceRedraw;
        (force update event for view)
      END;
    cPrinter:
      BEGIN
        SELF.SetCmd(cPrinter);
        SELF.fSampleView.ForceRedraw;
      END;
    OTHERWISE (always, so objects get a chance)
      DoMenuCommand := INHERITED DoMenuCommand(aCmdNumber);
  END; (Case)
END;

```

```

(-----)
($S ARes)
PROCEDURE TSampleDocument.SetCmd(CmdID: INTEGER);
BEGIN
  SELF.fDeviceCmd := CmdID; (save string in doc var)
END;

```

```

(-----)
($S ARes)
FUNCTION TSampleDocument.GetCmd: Integer;
BEGIN
  GetCmd := SELF.fDeviceCmd; (return variable as result)
END;

(=====)
($S AOpen)
PROCEDURE TSampleView.ISampleView(itsDocument: TSampleDocu-
ment);

```

```

VAR
  itsSize: VPoint;
BEGIN
  SetVPt(itsSize, 300, 300); (width, height: 32-bit numbers)
  SELF.IView(itsDocument, (doc that created view)
  NIL, (superview)
  gZeroVPt, (topLeft)
  itsSize, (width, height as a VPoint)
  sizeFixed, (width)
  sizeFixed); (height)
  SELF.fSampleDocument := itsDocument;
  (save address of document in view)
END;

```

```

(-----)
($S ARes)
PROCEDURE TSampleView.Draw(area: Rect); OVERRIDE;
(-----)
PROCEDURE MyDrawString(OutputStr:Str255);
Begin
  DrawString(OutputStr);
  vPos:=vPos+Leading;
  MoveTo(hPos,vPos);
End;

```

```

(-----)
PROCEDURE DrawMac;
CONST
  envMacIIcx = 6;
  envSE30 = 7; ( Not in the MPW 3.0 interfaces )
VAR

```

```

  MyConfigRecord: ConfigRecord; (from MAUtil.p)
  Item: Integer;
  bItem: Boolean;
  Str: Str255;
  aString: Str255;
  aStringHdl:StringHandle;
BEGIN
  aStringHdl := GetString(kMacStringID);
  FailNILResource(aStringHdl);
  Str:=aStringHdl^;
  MyDrawString(Str);

```

```

  DefineConfiguration(MyConfigRecord);
  (from MAUtil.p)

  Item:= MyConfigRecord.environsVersion;
  NumToString(Item, Str);
  Str:= Concat( ' environsVersion = ', Str );
  MyDrawString(Str);

```

```

CASE MyConfigRecord.machineType OF
  envMac:
    aString := 'Mac';
  envXL:
    aString := 'XL';
  envMachUnknown:
    aString := 'MachUnknown';
  env512KE:
    aString := '512KE';
  envMacPlus:
    aString := 'MacPlus';
  envSE:
    aString := 'SE';
  envMacII:
    aString := 'MacII';
  envMacIIX:
    aString := 'MacIIX';
  envMacIICx:
    aString := 'MacIICx';
  envSE30:
    aString := 'SE30';
  OTHERWISE
    aString := 'MachUnknown';

```

```

END;

Str:= Concat(' Machine Type = ' , aString);
MyDrawString(Str);

Item:= MyConfigRecord.systemVersion;
NumToString(Item, Str);
Str:= Concat(' systemVersion = ' , Str);
MyDrawString(Str);

CASE MyConfigRecord.processor OF
  envCPUUnknown:
    aString := 'CPUUnknown';
  env68000:
    aString := '68000';
  env68010:
    aString := '68010';
  env68020:
    aString := '68020';
  env68030:
    aString := '68030';
  OTHERWISE
    aString := 'CPUUnknown';
END;

Str:= Concat(' CPU Version = ' , aString);
MyDrawString(Str);

bItem:= MyConfigRecord.hasFPU;
IF bItem THEN Str:= ' has FPU = TRUE'
ELSE Str:= ' has FPU = FALSE';
MyDrawString(Str);

bItem:= MyConfigRecord.hasColorQD;
IF bItem THEN Str:= ' has Color QD = TRUE'
ELSE Str:= ' has Color QD = FALSE';
MyDrawString(Str);

CASE MyConfigRecord.keyboardType OF
  envUnknownKbd:
    aString := 'UnknownKbd';
  envMacKbd:
    aString := 'MacKbd';
  envMacAndPad:
    aString := 'MacAndPad';
  envMacPlusKbd:
    aString := 'MacPlusKbd';
  envAExtendKbd:
    aString := 'AExtendKbd';
  envStandADBKbd:
    aString := 'StandADBKbd';
  OTHERWISE
    aString := 'UnknownKbd';
END;

Str:= Concat(' Keyboard Type = ' , aString);
MyDrawString(Str);

Item:= MyConfigRecord.atDrvrsVersNum;
NumToString(Item, Str);
Str:= Concat(' Driver Version = ' , Str);
MyDrawString(Str);

Item:= MyConfigRecord.sysVRefNum;
NumToString(Item, Str);
Str:= Concat(' System Ref Number = ' , Str);
MyDrawString(Str);

bItem:= MyConfigRecord.hasROM128K;
IF bItem THEN Str:= ' has 128K ROM = TRUE'
ELSE Str:= ' has 128K ROM = FALSE';
MyDrawString(Str);

bItem:= MyConfigRecord.hasHFS;

```

```

IF bItem THEN Str:= ' has HFS = TRUE'
ELSE Str:= ' has HFS = FALSE';
MyDrawString(Str);

bItem:= MyConfigRecord.hasHierarchicalMenus;
IF bItem THEN Str:= ' has Hierarchical Menus = TRUE'
ELSE Str:= ' has Hierarchical Menus = FALSE';
MyDrawString(Str);

bItem:= MyConfigRecord.hasScriptManager;
IF bItem THEN Str:= ' has Script Manager = TRUE'
ELSE Str:= ' has Script Manager = FALSE';
MyDrawString(Str);

bItem:= MyConfigRecord.hasStyleTextEdit;
IF bItem THEN Str:= ' has StyleText Edit = TRUE'
ELSE Str:= ' has StyleTextEdit = FALSE';
MyDrawString(Str);

bItem:= MyConfigRecord.hasSoundManager;
IF bItem THEN Str:= ' has Sound Manager = TRUE'
ELSE Str:= ' has Sound Manager = FALSE';
MyDrawString(Str);

bItem:= MyConfigRecord.hasWaitNextEvent;
IF bItem THEN Str:= ' has WaitNextEvent = TRUE'
ELSE Str:= ' has WaitNextEvent = FALSE';
MyDrawString(Str);

bItem:= MyConfigRecord.hasSCSI;
IF bItem THEN Str:= ' has SCSI = TRUE'
ELSE Str:= ' has SCSI = FALSE';
MyDrawString(Str);

bItem:= MyConfigRecord.hasDesktopBus;
IF bItem THEN Str:= ' has Desktop Bus = TRUE'
ELSE Str:= ' has Desktop Bus = FALSE';
MyDrawString(Str);

```

END;

(-----)

PROCEDURE DrawPrinter;

TYPE

PrGeneralPtr = ^TGetRs1B1k;

CONST

GetRs1Data = 4;

VAR

aStringHdl:StringHandle;

Str: Str255;

Item: LongInt;

pDialog: PrGeneralPtr;

i,NumberOfRecords: Integer;

BEGIN

aStringHdl := GetString(kPrinterStringID);

FailNILResource(aStringHdl);

Str:=aStringHdl^;

MyDrawString(Str);

pDialog:= PrGeneralPtr(NewPtr(SIZEOF(TGetRs1B1k)));

PrOpen;

IF PrError = noErr then

BEGIN

pDialog^.iOpCode:= GetRs1Data;

PrGeneralPtr(pDialog);

IF PrError = noErr then

BEGIN

NumberOfRecords:= pDialog^.iRs1RecCnt;

FOR i:= 1 to NumberOfRecords DO

BEGIN

Item := pDialog^.rgRs1Rec[i].iXRsl;

NumToString(Item, Str);

Str:= Concat(' X Resolution = ' , Str);

MyDrawString(Str);

Item := pDialog^.rgRs1Rec[i].iYRsl;

```

        NumToString(Item, Str);
        Str:= Concat(' Y Resolution = ', Str);
        MyDrawString(Str);
    END;
END
ELSE
BEGIN
    Item:= LongInt(PrError);
    NumToString(Item, Str);
    Str:= Concat(' PrGeneral Error = ', Str);
    MyDrawString(Str);
END
END
ELSE
BEGIN
    Item:= LongInt(PrError);
    NumToString(Item, Str);
    Str:= Concat(' PrOpen Error = ', Str);
    MyDrawString(Str);
END;
PrClose;
IF PrError = noErr then
BEGIN
END
ELSE
BEGIN
    Item:= LongInt(PrError);
    NumToString(Item, Str);
    Str:= Concat(' PrClose Error = ', Str);
    MyDrawString(Str);
END;
END;
(-----)
VAR
    itsQDExtent: Rect;
    FontStr:      Str255;
    FontNumber:   Integer;
    aStringHdl:   StringHandle;
    Mode:         Integer;
BEGIN
    PenNormal; (in case someone else changed it)
    PenSize(1, 1);
    PenPat(black);

    SELF.GetQDExtent(itsQDExtent); (get size of this view)
    InsetRect(itsQDExtent, 2, 2);
    FrameRect(itsQDExtent); (draw an enclosing rect)

    aStringHdl := GetString(kFontStringID);
    FailNILResource(aStringHdl);
    FontStr := aStringHdl^; (save string in variable)
    GetFNum(FontStr, FontNumber);

    TextMode(srcOr);
    TextFont(FontNumber);
    TextFace(1);
    hPos:=10;
    vPos:=32;
    MoveTo(hPos, vPos);
    TextSize(12);
    Leading:=14;

    Mode:=SELF.fSampleDocument.GetCmd;
    CASE Mode OF
        cMac:
            DrawMac;
        cPrinter:
            DrawPrinter;
        OTHERWISE;
    END;

    PenNormal;
END;

```

File Samples.r:

```

#ifndef __MacAppTypes__
#include "MacAppTypes.r"
#endif

#include "SysTypes.r"

#ifdef qDebug
include "Debug.rsrc";
#endif
include "MacApp.rsrc";
include "Printing.rsrc";

include "Sample" 'CODE'; /* include CODE resources from
compile/link */

/* cAboutApp, cNew, etc. defined in
"(MARIncludes)MacAppTypes.r" */
#define kWindRsrcID 1001
#define kPrinterStringID 1001 /* STR resource ID */
#define kMacStringID 1002 /* STR resource ID */
#define kFontStringID 1003
#define cMac 2001 /* menu command number */
#define cPrinter 2002 /* menu command number */

resource 'vers' (1) (
    0x00, 0x95, beta, 0x2, verUS,
    "0.95b2",
    "0.95 b2 (US), ©1989 MacTutor"
);

resource 'vers' (2) (
    0x05, 0x08, development, 0x02, verUS,
    "Vol. 5 No. 8",
    "MacTutor, Vol. 5 No. 8"
);

resource 'STR' (kPrinterStringID) (
    "This is the printer (Driver)";
);

resource 'STR' (kMacStringID) (
    "This is the Mac";
);

resource 'STR' (kFontStringID) (
    "times";
);

resource 'cmnu' (1) (
    1,
    textMenuProc,
    0x7FFFFFFD,
    enabled,
    apple,
    (
        "About Sample...", noIcon, noKey, noMark, plain, cAboutApp;
        "-", noIcon, noKey, noMark, plain, noCommand
    )
);

resource 'cmnu' (2) (
    2,
    textMenuProc,
    0x7FFFBBB,
    enabled,
    "File",
    (
        "New", noIcon, "N", noMark, plain, cNew;
        "Open...", noIcon, "O", noMark, plain, cOpen;
    )
);

```

```

    "-", noIcon, noKey, noMark, plain, nocommand;
    "Close", noIcon, "W", noMark, plain, cClose;
    "Save", noIcon, "S", noMark, plain, cSave;
    "Save As...", noIcon, noKey, noMark, plain, cSaveAs;
    "Save a Copy In...", noIcon, noKey, noMark, plain, cSave-
Copy;
    "-", noIcon, noKey, noMark, plain, nocommand;
    "Page Setup...", noIcon, noKey, noMark, plain, cPageSetup;
    "Print One", noIcon, noKey, noMark, plain, cPrintOne;
    "Print...", noIcon, "P", noMark, plain, cPrint;
    "-", noIcon, noKey, noMark, plain, nocommand;
    "Quit", noIcon, "Q", noMark, plain, cQuit
}
);

resource 'cmnu' (3) {
    3,
    textMenuProc,
    0x7FFFFFBD,
    enabled,
    "Edit",
    {
        "Undo", noIcon, "Z", noMark, plain, cUndo;
        "-", noIcon, noKey, noMark, plain, nocommand;
        "Cut", noIcon, "X", noMark, plain, cCut;
        "Copy", noIcon, "C", noMark, plain, cCopy;
        "Paste", noIcon, "V", noMark, plain, cPaste;
        "Clear", noIcon, noKey, noMark, plain, cClear;
        "-", noIcon, noKey, noMark, plain, nocommand;
        "Show Clipboard", noIcon, noKey, noMark, plain, cShowClip-
board
    }
};

resource 'cmnu' (4) {
    4,
    textMenuProc,
    0x7FFFFFBD,
    enabled,
    "Sample",
    {
        "Query Mac", noIcon, "M", noMark, plain, cMac;
        "-", noIcon, noKey, noMark, plain, nocommand;
        "Query Printer", noIcon, "P", noMark, plain, cPrinter;
    }
};

resource 'MBAR' (128) {
    (1; 2; 3; 4)
};

resource 'WIND' (kWindRsrcID, purgeable) {
    (50, 40, 375, 450),
    zoomDocProc,
    invisible,
    goAway,
    0x0,
    "<<<<>>>>"
};

resource 'SIZE' (-1) {
    saveScreen,
    acceptSuspendResumeEvents,
    enableOptionSwitch,
    canBackground,
    MultiFinderAware,
    backgroundAndForeground,
    dontGetFrontClicks,
    ignoreChildDiedEvents,
    is32BitCompatible,
    reserved, reserved, reserved, reserved, reserved, reserved,
    reserved,
    #if qDebug
    340 * 1024,

```

```

    320 * 1024
    #else
    150 * 1024,
    120 * 1024
    #endif
};

resource 'seg!' (256,
    #if qNames
    "Sample",
    #endif
    purgeable) {
    { "GTerminate";
      "GNonRes";
      "GOpen";
      "GWriteFile";
      "GFile";
      "GClose";
      "GClipboard";
      "GDoCommand";
      "MAView";
      "GSelectCommand";
    }
};

resource 'mem!' (256,
    #if qNames
    "Sample",
    #endif
    purgeable) {
    43 * 1024, /* Add to temporary reserve */
    0, /* Add to permanent reserve */
    0 /* Add to stack space */
};

resource 'DITL' (201, purgeable) {
    /* array DITLArray: 3 elements */
    /* [1] */
    (130, 182, 150, 262),
    Button {
        enabled,
        "OK"
    };
    /* [2] */
    (10, 80, 110, 270),
    StaticText {
        disabled,
        "This program queries the printer & Mac"
        "\n\nThis program was written "
        "with MacApp 2.0b9 © © 1985-1989 Apple Computer, Inc."
    };
    /* [3] */
    (10, 20, 42, 52),
    Icon {
        disabled,
        1
    }
}
};

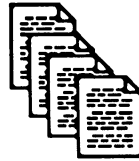
resource 'ALRT' (201, purgeable) {
    (90, 100, 250, 412),
    201,
    {
        OK, visible, silent;
        OK, visible, silent;
        OK, visible, silent;
        OK, visible, silent
    }
};

```



TechNotes

New CDEF Messages



James Plamondon
Berkeley, CA
Volume 5, Number 8

System 7.0 will present all Macintosh programmers with new opportunities and new challenges. One of the challenges will be making your code 32-bit clean. This can be particularly difficult if your code uses parts of the Toolbox that are not 32-bit clean. Some of the routines in the Control Manager, for example, are at best only 31-bit clean — that's close, but not close enough for System 7.0. This article will answer the challenge by explaining both the problem with, and the solution to, the Control Manager's unclean habits.

The specific problem that this article addresses lies in the calcCRgns CDEF message sent to all CDEFs. The message is sent to a control's CDEF to calculate the control's enclosing region, or the enclosing region of its indicator (if any). In Inside Macintosh [IM v1 p330], the calcCRgns CDEF message is explained as follows: "Param is a QuickDraw region handle.... If the high-order bit of param is set, the region requested is that of the control's indicator rather than the control as a whole. The definition function should clear the high *byte* (not just the high bit) of the region handle before attempting to update the region."

You can see that it's pretty hard to be 32-bit clean when you're supposed to go around clearing bytes in handles. The handle could contain a perfectly valid 32-bit address; clearing the high byte would change the address to something entirely different — and accessing or changing anything via that address would be a terrible mistake. Yet, it is required by the Control Manager, so what is a programmer to do?

First, one can at least improve the situation by not clearing the whole high byte, as directed. Instead, just interrogate the high bit to see what region needs to be calculated, and then clear it — leaving the rest of the bits alone. This makes the code 31-bit clean; that's not good enough for System 7.0, but it'll do for the old systems.

For System 7.0, Apple has defined two new CDEF messages: calcCntlRgn (10) and calcThumbRgn (11). Instead of sending a calcCRgns message and having the CDEF figure out whether the system wants the control region or the indicator (thumb) region, the system will figure out which it wants, and send the CDEF the appropriate (new) message. The CDEF can then treat the region handle like a region handle, instead of like a secret message that has to be decoded.

To make sure the world is a happy place, and that backward compatibility is maintained, a CDEF should always implement both the calcCRgns message and the new CDEF messages. That way, whether the CDEF finds itself in use under either System 7.0 or a pre-7.0 system (which is pretty likely for the next year or two), it will respond in the manner the system expects.

There are a couple of related issues. First, a control's variation code used to be stored in the high byte of the control

record's contrlDefProc field. To get the variation code, the programmer would mask it out of the high byte. This is unclean, since the contrlDefProc field may contain a valid 32-bit address. A new routine, GetCVariant(), was provided a couple of years ago (IM v5 p222) to return the control's variation code. Be sure to use it.

Also, whenever writing custom CDEFs or/and other DEFs, remember that The Management Reserves the Right to Add New Messages at Any Time. Therefore, be sure that if your DEF receives a message it doesn't recognize, it just returns what it was given without doing anything (and without blowing up).

Lastly, if you're going to add any custom messages to your custom DEF, set the new message numbers to constants greater than 128. This gives Apple the freedom it needs to add new messages, without tromping on your custom messages.

That's about all the discussion this issue will bear. See the revision to Tech Note #212 for the official version of this same information. Below, I have appended the old, unclean calcCRgns implementation, the new and improved 31-bit clean calcCRgns implementation, and a shell CDEF entry-point function that shows how all these messages should be handled. Note that the 'param' actual argument is a valid handle to an empty region, so the content of the region indicated by param needs to be changed, not the value of param itself. You couldn't make any change to param stick anyway, since it's not a 'var' parameter (i.e., the CDEF gets a copy of param's value, not its address). Also, I'm assuming the existence of some routines, such as MyCalcThumbRgn() and MyCalcCtlRgn(), that are different for each different CDEF, and which are therefore not given here.

I'd like to thank Andrew Shebanow, Larry Tesler, Charlie Oppenheimer, and Steve Goldberg at Apple — way to go, guys!

```
(*****  
MyControl: Main entry point. Call appropriate  
message-handling routine.  
*****)
```

```
FUNCTION MyControl(varCode: INTEGER;  
theCntl: ControlHandle;  
message: INTEGER;  
param: LONGINT): LONGINT;
```

```
BEGIN
```

```
MyControl := 0; ( default )
```

```
CASE message OF
```

```
drawCntl:  
doDrawCntl(theCntl, varCode, param);  
testCntl:  
MyControl := doTestCntl(theCntl, param);  
initCntl:  
doInitCntl(theCntl, varCode);  
dispCntl:  
doDispCntl(theCntl, varCode);
```

```

autoTrack:
    doAutoTrack(theCntrl, varCode, param);

calcCRgns: ( support for pre-7.0 )
    doCalcCRgnsNew(theCntrl, param);
calcCntrlRgn: ( new in System 7.0 )
    MyCalcCntrlRgn(theCntrl, RgnHandle(param));
calcThumbRgn: ( new in System 7.0 )
    MyCalcThumbRgn(theCntrl, RgnHandle(param));

```

OTHERWISE

```

    ; ( ignore all other messages )
END; ( case )
END; ( MyControl )

```

```

(*****
doCalcCRgnsOld: Calculate the region the
control or its indicator occupies in its
window. Old method: Only 24-bit clean
- don't use this method anymore; use
doCalcCRgnsNew(), below.
*****)

```

```

PROCEDURE doCalcCRgnsOld(theCntrl: ControlHandle;
    param: LONGINT);

```

```

BEGIN
    if (param < 0) then ( high bit is sign bit )
        begin
            ( clear high BYTE and set thumb rgn )
            param := BitAnd(param, $0FFFFFFF);
            MyCalcThumbRgn(theCntrl, RgnHandle(param));
        end
    end

```

```

    else
        begin
            ( clear high BYTE and set control rgn )
            param := BitAnd(param, $0FFFFFFF);
            MyCalcCntrlRgn(theCntrl,
                RgnHandle(param));
        end;
END; ( doCalcCRgnsOld )

```

```

(*****
doCalcCRgnsNew: New method: Only 31-bit
clean, but that's the best we can do.
*****)

```

```

PROCEDURE doCalcCRgnsNew(theCntrl: ControlHandle;
    param: LONGINT);

```

```

BEGIN
    if (param < 0) then ( high bit is sign bit )
        begin
            ( clear high BIT and set thumb rgn )
            param := BitAnd(param, $7FFFFFFF);
            MyCalcThumbRgn(theCntrl, RgnHandle(param));
        end
    else
        begin
            ( clear high BIT and set control's rgn )
            param := BitAnd(param, $7FFFFFFF);
            MyCalcCntrlRgn(theCntrl, RgnHandle(param));
        end;
END; ( doCalcCRgnsNew )

```



Pascal Procedures

A Look At The PREC Resource



Dave Kelly
MacTutor Editorial Board
Volume 5, Number 8

As we all know the Apple ImageWriter printer has taken a back seat over the past several years to the LaserWriter. Most of us prefer Laser quality printing to that of the ImageWriter. However, another reality is the extra cost of the LaserWriter.

A quick look back in history reminds us that there are some provisions in the ImageWriter drivers which allow applications to "own" their own page sizes. This facility is brought to use by way of the PREC resource. In particular, the driver itself uses PREC resource ID=3 for the page size definitions as a default. If an application contains a PREC resource with ID=4 these size definitions take precedence. This may be old news to most of us, but it doesn't seem to be well documented anywhere. The only place I found information about this was in some old notes from April, 1985. The Macintosh Technical notes don't give any indication about this subject.

A sample RMaker source defining a PREC 4 resource that encodes the default configuration of the Page Setup Dialog is:

```
Type PREC = GNRL
*Paper size info - localizable, customizable.
    ,4
    .1
*Number of paper-size buttons used (max is 6):
5
*6 paper sizes in 120ths (decimal!):
*1st number is vertical, 2nd is horizontal.
*First button: 8.5 X 11" 'US letter' paper
1320
1020
*Second button: 8.25 X 11.66" 'A4 letter'
1400
990
*Third button: 8.5 X 14" 'US legal' paper
1680
1020
*4th button: 8.25 X 12" 'international fanfold'
1440
990
*Fifth button: 14 X 11" 'computer paper'
1320
1680
*Sixth button: - not visible
0
0
.P
*Titles for six buttons.
US Letter
A4 Letter
US Legal
International Fanfold
Computer Paper
?
```

*The blank line just above this one is necessary!

One easy way of creating this type of resource is with RMaker. It may not be very convenient for non-programmers to create the resource without help. The Pascal application included

here allows editing and creation of the PREC 4 resource. Handling the resource is not too difficult although there are a couple of things you need to know to manipulate the structure.

1. The type PREC is reserved for use by the Macintosh print driver. It should not be used in any other way than described here and not with any other resource ID other than 4. The PREC 3 resource in the printer driver itself could be modified although is not recommended.

2. Exactly six buttons must be defined in the PREC 4 resource even if you are not using all the buttons. One character titles should be used for unused buttons.

3. All PREC 4 numbers are in decimal.

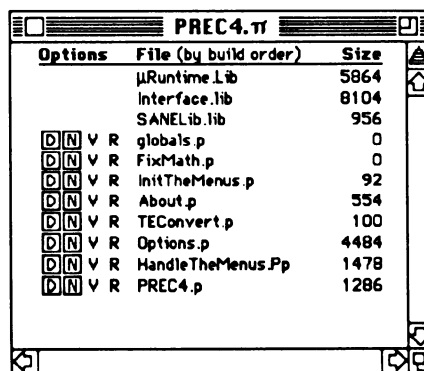
4. The first item in the PREC resource is the number of buttons to be displayed, a word-size integer. Whatever number is used here will be used by the printer driver to determine the number of items in the Page Setup dialog. If you use a number greater than six you are in big trouble.

5. The dimensions follow next as an array of six pairs of integers. Each pair of numbers corresponds to the paper sizes with the first integer in each pair as the height and the second integer as the width. These numbers are given in 120ths of an inch.

6. The remainder of the data is presented as an array of variable length Pascal strings. To get to each string, these strings need to be handled as a byte or char array. The first byte is the length of the first string which is followed by the first string, followed by the length of the second string, and so forth.

I credit my assistant, Prototyper for the shell of this program with the necessary additions added for loading, saving and manipulating the PREC 4 resource. The List Manager is used to display and select the 6 possible buttons.

With all this said and done, it is good to note that Apple has announced that with System 7.0, spooling will be supported for all printers, not just the LaserWriter. Personally, I use the AppleTalk ImageWriter driver with 2 Macs wishing that the background printing was supported with the current system.



```

unit globals;

interface
var
  MyWindow: WindowPtr; (Window pointer)
  TitleFieldhnd1: TEHandle; (Text Edit field handle)
  WidthFieldhnd1: TEHandle; (Text Edit field handle)
  HeightFieldhnd1: TEHandle; (Text Edit field handle)
  reply: SFRReply;
type
  realprec = record
    numberofbuttons: signedbyte;
    Size: array[1..6] of point;
    title: packed array[0..255] of char;
  end;
  realprec4ptr = ^realprec;
  realprec4hnd1 = ^realprec4ptr;
  prec = record
    numberofbuttons: integer;
    Size: array[1..6] of point;
    Title: array[1..6] of str255;
  end;
  prec4ptr = ^prec;
  prec4hnd1 = ^prec4ptr;
var
  therealprec4: realprec;
  therealprec4hnd1: realprec4hnd1;
  prec4: prec;
  theprec4hnd1: prec4hnd1;
implementation
end.

```

```

unit InitTheMenus;
(File name: InitTheMenus.p)
(Function: Pull in menu lists from a resource file.)
( This procedure is called once at program start.)
( AppleMenu is the handle to the Apple menu, it is also)
( used in the procedure that handles menu events.)
(History: 6/10/89 Original by Prototyper. )

```

```

interface
procedure Init_My_Menus; (Initialize the menus)
var
  AppleMenu: MenuHandle; (Menu handle)
  M_File: MenuHandle; (Menu handle)
  M_Edit: MenuHandle; (Menu handle)
implementation
procedure Init_My_Menus; (Initialize the menus)
const
  Menu1 = 1001; (Menu resource ID)
  Menu2 = 1002; (Menu resource ID)
  Menu3 = 1003; (Menu resource ID)
begin
  (Start of Init_My_Menus)
  ClearMenuBar; (Clear any old menu bars)
  AppleMenu := GetMenu(Menu1); (Get the menu)
  InsertMenu (AppleMenu,0); (Insert this menu)
  AddResMenu(AppleMenu,'DRVr'); (Add in DAs)
  M_File := GetMenu(Menu2); (Get the menu)
  InsertMenu (M_File,0); (Insert this menu)
  M_Edit := GetMenu(Menu3); (Get the menu)
  InsertMenu (M_Edit,0); (Insert this menu)
  DrawMenuBar; (Draw the menu bar)
end; (End of procedure Init_My_Menus)
end. (End of this unit)

```

```

unit About;
(File name: About.p )

```

```

(Function: Handle a dialog)
(History: 6/10/89 Original by Prototyper. )

```

```

interface
procedure D_About;
implementation
const (These are item numbers for controls in the Dialog)
  I_OK = 1;
  I_x = 2;
  I_Icon6 = 3;
var
  ExitDialog: boolean; (Flag used to exit the Dialog)
  DoubleClick: boolean; (Flag a double click on list)
  MyPt: Point; (Current list selection point)
  MyErr: OSErr; (OS error returned)

(=====)
function MyFilter (theDialog: DialogPtr; var theEvent:
EventRecord; var itemHit: integer): boolean;
var
  tempRect: Rect;
begin
  MyFilter := FALSE;
  if (theEvent.what = MouseDown) then (Only do on a mouse
click)
  begin
    MyPt := theEvent.where; (Get the point where the mouse
was clicked)
    GlobalToLocal(MyPt); (Convert global to local)
  end;
end;

(=====)
procedure D_About;
var
  GetSelection: DialogPtr; (Pointer to this dialog)
  tempRect: Rect; (Temporary rectangle)
  DType: Integer; (Type of dialog item)
  Index: Integer; (For looping)
  DItem: Handle; (Handle to the dialog item)
  CItem, CTempItem: controlhandle; (Control handle)
  sTemp: Str255; (Get text entered, temp holding)
  itemHit: Integer; (Get selection)
  temp: Integer; (Get selection, temp holding)
  dataBounds: Rect; (Rect to setup the list)
  cSize: Point; (Pointer to a cell in a list)
  Icon_Handle: Handle; (Temp handle to read an Icon into)
  NewMouse: Point; (Mouse loc during tracking Icon)
  InIcon: boolean; (Flag to say pressed in an Icon)
  ThisEditText: TEHandle; (Handle to get Dialogs TE rec)
  TheDialogPtr: DialogPeek; (Ptr to Dialogs definition
record, contains the TE record)
  (This is an update routine for non-controls in the dialog)
  (This is executed after the dialog is uncovered by an alert)
procedure Refresh_Dialog; (Refresh dialogs non-controls)
var
  rTempRect: Rect; (Temp rectangle used for drawing)
begin
  SetPort(GetSelection); (Point to our dialog window)
  GetDItem(GetSelection, I_OK, DType, DItem,
tempRect); (Get the item handle)
  PenSize(3, 3); (Change pen to draw default outline)
  InsetRect(tempRect, -4, -4);
  FrameRoundRect(tempRect, 16, 16); (Draw the outline)
  PenSize(1, 1); (Restore pen size to the default value)
end;
begin (Start of dialog handler)

```

```

GetSelection := GetNewDialog(2, nil, Pointer(-1));
(Bring in the dialog resource)
ShowWindow(GetSelection); (Open a dialog box)
SelectWindow(GetSelection); (Lets see it)
SetPort(GetSelection); (Prepare to add conditional text)
TheDialogPtr := DialogPeek(GetSelection);
ThisEditText := TheDialogPtr^.textH; (Get to the TE record)
HLock(Handle(ThisEditText)); (Lock it for safety)
ThisEditText^.txSize := 12; (TE Point size)
TextSize(12); (Window Point size)
ThisEditText^.txFont := systemFont; (TE Font ID)
TextFont(systemFont); (Window Font ID)
ThisEditText^.txFont := 0; (TE Font ID)
ThisEditText^.fontAscent := 12; (Font ascent)
ThisEditText^.lineHeight := 12 + 3 + 1; (Font ascent +
descent + leading)
HUnlock(Handle(ThisEditText)); (Unlock handle when done)
(Setup initial conditions)
RefreshDialog;
ExitDialog := FALSE; (Do not exit dialog handle loop yet)
repeat (Start of dialog handle loop)
    ModalDialog(nil, itemHit); (Wait until an item is hit)
    GetDItem(GetSelection, itemHit, DType, DItem,
tempRect); (Get item information)
    CItem := Pointer(DItem); (Get the control handle)
    (Handle it real time)
    if (ItemHit = I_OK) then
        begin
            (?? Code to handle this button goes here)
            ExitDialog := TRUE;
            RefreshDialog;
        end; (End for this item selected)
    if (ItemHit = I_Icon6) then (Handle the Icon)
        begin
            Icon_Handle := GetIcon(10032);
            if (Icon_Handle <> nil) then
                begin (Get ready to plot the hilighted icon)
                    EraseRect(tempRect); (Erase the original icon)
                    PlotIcon(tempRect, Icon_Handle);
                end; (End of drawing hilighted icon)
                InIcon := TRUE; (Flag as mouse in the Icon)
                repeat (Start of mouse tracking routine)
                    GetMouse(NewMouse); (Get latest mouse position)
                    if (PtInRect(NewMouse, tempRect)) then
                        begin (...yes, over the Icon)
                            if not (InIcon) then
                                begin (...yes, so it is unhilighted)
                                    Icon_Handle := GetIcon(10032);
                                    if (Icon_Handle <> nil) then
                                        begin (...yes, have hilighted Icon)
                                            EraseRect(tempRect);
                                            PlotIcon(tempRect,
Icon_Handle); (Draw the hilighted icon)
                                        end; (End of Draw the hilighted icon)
                                        InIcon := TRUE;
                                    end; ()
                                end
                            else if InIcon then (Then draw unhilighted Icon)
                                begin (Start drawing unhilighted Icon)
                                    Icon_Handle := GetIcon(32);
                                    if (Icon_Handle <> nil) then
                                        begin (...yes, have hilighted Icon)
                                            EraseRect(tempRect); (Erase hilited icon)
                                            PlotIcon(tempRect, Icon_Handle); (Draw
the standard icon)
                                        end; (End of Draw the icon)
                                        InIcon := FALSE;

```

```

        end; ()
    until not (StillDown);
    if (PtInRect(NewMouse, tempRect)) then
        begin (...yes, released in the Icon)
            RefreshDialog;
        end; (End of released in the Icon)
        Icon_Handle := GetIcon(32);
        if (Icon_Handle <> nil) then
            begin (...yes, have hilighted Icon)
                EraseRect(tempRect); (Erase the hilighted icon)
                PlotIcon(tempRect, Icon_Handle);
            end; (End of Draw the icon)
        end; (End for this item selected)
    until ExitDialog; (Handle dialog items until exit)
    (Get results after dialog)
    DisposDialog(GetSelection); (Flush dialog out of memory)
end; (End of procedure)
end. (End of unit)

```

```

unit TEConvert;
(General utilities for conversion between TextEdit and
strings)
( see Macintosh Technical note #18 )
interface
( uses)
( MemTypes, QuickDraw, OSIntf, ToolIntf; )
    procedure TERecToStr (hTE: TEHandle; var str: Str255);
    (TERecToStr converts the TextEdit record hTE to string str.)
    (If necessary, the text will be truncated to 255 characters.)
    procedure StrToTERec (str: Str255; hTE: TEHandle);
    (StrToTERec converts the string str to TextEdit record hTE. )
implementation
    procedure TERecToStr (hTE: TEHandle; var str: Str255);
    begin
        GetIText(hTE^.hText, str);
    end;
    procedure StrToTERec (str: Str255; hTE: TEHandle);
    begin
        TESSetText(POINTER(ORD4(@str) + 1), ORD4(length(str)),
hTE);
    end;
end.

```

```

unit Options;
(File name: Options.p)
(Function: Handle a Window)
(History: 6/10/89 Original by Prototyper. )
interface
    uses
        globals, FixMath, TEConvert;
    procedure preparewidth (i: integer);
    procedure prepareheight (i: integer);
    procedure SetFields;
    procedure SetDefaultPREC;
    (Initialize us so all our routines can be activated)
    procedure Init_Options;
    (Record status of all edit fields & buttons and update list)
    procedure Recordstatus;
    (Close our window)
    procedure Close_Options (whichWindow: WindowPtr; var
theInput: TEHandle);
    (Open our window and draw everything)
    procedure Open_Options (var theInput: TEHandle);
    (Update our window, someone uncovered a part of us)
    procedure Update_Options (whichWindow: WindowPtr);
    (Handle action to our window, like controls)
    procedure Do_Options (myEvent: EventRecord; var theInput:

```

```
TEHandle);
implementation
const
  TitleFieldid = 23;           (Edit text ID)
  WidthFieldid = 24;           (Edit text ID)
  HeightFieldid = 25;          (Edit text ID)
var
  tempRect, temp2Rect: Rect; (Temporary rectangle)
  Index: Integer;             (For looping)
  CtrlHandle: ControlHandle;  (Control handle)
  sTemp, sTemp1: Str255; (Get text entered, temp holding)
  List_I_Titles: ListHandle; (List Manager list handle)
  Rect_I_Titles: Rect; (List Manager list rectangle)
  dataBounds: Rect; (For use by lists)
  cSize: Point; (For use by lists)
  theRow: integer; (Used for adding rows to lists)
  num: extended;
  fixednum: Fixed;
  theCell: Cell;

(=====)
procedure SetTeWidth (i: integer);
begin
  num := PREC4.size[i].h / 120;
  fixednum := X2Fix(num);
  NumToString(HiWord(Fixednum), sTemp);
  num := (num - HiWord(Fixednum)) * 100;
  fixednum := X2Fix(num);
  NumToString(HiWord(Fixednum), sTemp1);
  sTemp := concat(sTemp, '.', sTemp1);
  TEText(Pointer(Ord4(@sTemp) + 1), length(sTemp),
WidthFieldhnd1); (Place default text in the TE area)
  SetRect(TempRect, 220, 82, 270,
102); (left,top,right,bottom of WidthField Rectangle)
  TEUpdate(TempRect, WidthFieldhnd1);
end;

(=====)
procedure SetTeHeight (i: integer);
begin
  num := PREC4.size[i].v / 120;
  fixednum := X2Fix(num);
  NumToString(HiWord(Fixednum), sTemp);
  num := (num - HiWord(Fixednum)) * 100;
  fixednum := X2Fix(num);
  NumToString(HiWord(Fixednum), sTemp1);
  sTemp := concat(sTemp, '.', sTemp1);
  TEText(Pointer(Ord4(@sTemp) + 1), length(sTemp),
HeightFieldhnd1); (Place default text in the TE area)
  SetRect(TempRect, 326, 82, 376,
102); (left,top,right,bottom of HeightField Rectangle)
  TEUpdate(TempRect, HeightFieldhnd1);
end;

(=====)
procedure SetFields;
var
  i, datalen: integer;
begin
  for i := 6 downto 1 do
    begin
      datalen := length(PREC4.Title[i]);
      cSize.v := i - 1;
      LSetCell(Pointer(Ord4(PREC4.Title[i]) + 1), datalen,
cSize, List_I_Titles); (Set string in row )
      LSetSelect(false, cSize, List_I_Titles);
    end;

```

```
SetTeWidth(1);
SetTeHeight(1);
TEText(Pointer(Ord4(PREC4.Title[1]) + 1),
length(PREC4.Title[1]), TitleFieldhnd1);
(default text in the TE area)
SetRect(TempRect, 220, 40, 380, 62);
TEUpdate(TempRect, TitleFieldhnd1);
LSetSelect(true, cSize, List_I_Titles);
end;

```

```
(=====)
procedure SetDefaultPREC;
begin
  with prec4 do
    begin
      numberofbuttons := 5;
      Size[1].v := 1320; ( 11.0" )
      Size[1].h := 1020; ( 8.50" )
      Size[2].v := 1400; ( 11.66" )
      Size[2].h := 990; ( 8.25" )
      Size[3].v := 1680; ( 14.0" )
      Size[3].h := 1020; ( 8.50" )
      Size[4].v := 1440; ( 14.0" )
      Size[4].h := 990; ( 8.50" )
      Size[5].v := 1320; ( 14.0" )
      Size[5].h := 1680; ( 8.50" )
      Size[6].v := 0; ( undefined)
      Size[6].h := 0; ( undefined)
      Title[1] := 'US Letter';
      Title[2] := 'A4 Letter';
      Title[3] := 'US Legal';
      Title[4] := 'International Fanfold';
      Title[5] := 'Computer Paper';
      Title[6] := '?';
    end;
    setFields;
    SetWTitle(MyWindow, 'Page Option Defaults');
  end;

(=====)
  (Initialize us so all our routines can be activated)
  procedure Init_Options;
  begin
    (Start of Window initialize routine)
    MyWindow := nil;
    (Make sure other routines know we are not valid yet)
  end;
  (End of procedure)

(=====)
  (Record status of all edit fields & buttons and update list)
  procedure Recordstatus;
  var
    i, datalen: integer;
    tempnumber: real;
  function strtoc (s: str255): extended;
  var
    number: longint;
    total, multiplier: real;
    decimalposition, i: integer;
  begin
    total := 0;
    decimalposition := pos('.', s);
    multiplier := 1;
    for i := (decimalposition - 1) downto 1 do
      begin
        stringtonum(s[i], number);
        total := total + (number * multiplier);
        multiplier := multiplier * 10;
      end;

```

```

end;
multiplier := 0.1;
for i := (decimalposition + 1) to length(s) do
begin
    stringtonum(s[i], number);
    total := total + (number * multiplier);
    multiplier := multiplier / 10;
end;
if decimalposition = 0 then
begin
    multiplier := 1;
    total := 0;
    for i := length(s) downto 1 do
    begin
        stringtonum(s[i], number);
        total := total + (number * multiplier);
        multiplier := multiplier * 10;
    end;
end;
strtox := total;
end;
begin
    Csize.h := 0;
    TRectToStr(TitleFieldhdl, sTemp);
    PREC4.Title[cSize.v + 1] := sTemp;
    dataLen := length(sTemp);
    LSetCell(Pointer(ord(sTemp) + 1), dataLen, cSize,
List_I_Titles); (Set string in row )
    TRectToStr(WidthFieldhdl, sTemp);
    i := 120;
    tempnumber := strtox(sTemp) * i;
    PREC4.size[cSize.v + 1].h := trunc(tempnumber);
    TRectToStr(HeightFieldhdl, sTemp1);
    tempnumber := strtox(sTemp1) * i;
    PREC4.size[cSize.v + 1].v := trunc(tempnumber);
end;

(=====)
(Close our window)
procedure Close_Options;
begin          (Start of Window close routine)
    if (MyWindow <> nil) and ((MyWindow = whichWindow) or
ord4(whichWindow) = -1)) then
        begin
            if (theInput = TitleFieldhdl) then
                theInput := nil; (Clear the handle used)
            if (theInput = WidthFieldhdl) then
                theInput := nil; (Clear the handle used)
            if (theInput = HeightFieldhdl) then
                theInput := nil; (Clear the handle used)
            DisposeWindow(MyWindow); (Clear window and controls)
            MyWindow := nil;
        end;          (End for if (MyWindow<>nil))
    end;          (End of procedure)

(=====)
(Update our window, someone uncovered a part of us)
procedure UpDate_Options;
var
    SavePort: WindowPtr; (Place to save the last port)
    sTemp: Str255; (Temporary string)
begin          (Start of Window update routine)
    Recordstatus;
    if (MyWindow <> nil) and (MyWindow = whichWindow)
then (Handle an open when already opened)
        begin
            GetPort(SavePort); (Save the current port)

```

```

SetPort(MyWindow); (Set the port to my window)
TextFont(systemFont); (Set the font to draw in)
(Draw a string of text, )
SetRect(tempRect, 273, 85, 318, 102);
sTemp := 'X';
TextBox(Pointer(ord(sTemp) + 1), length(sTemp),
tempRect, teJustLeft);
TextFont(applFont); (Set default application font)

TextFont(systemFont); (Set the font to draw in)
(Draw a string of text, )
SetRect(tempRect, 378, 85, 423, 102);
sTemp := ' ';
TextBox(Pointer(ord(sTemp) + 1), length(sTemp),
tempRect, teJustLeft);
TextFont(applFont);
(Update a TE box, US Letter )
SetRect(TempRect, 220, 40, 380,
65); (left,top,right,bottom)
FrameRect(TempRect); (Frame this TE area)
InsetRect(TempRect, 3, 3); (Surround the TE area)
if (TitleFieldhdl <> nil) then
    begin          (Update the TE area)
        TEUpdate(TempRect, TitleFieldhdl);
        (Update the TE area)
        TextFont(applFont);
        (Set the default application font)
    end;          (End of the TE area)
(Update a TE box, Width )
SetRect(TempRect, 220, 82, 270, 102);
FrameRect(TempRect); (Frame this TE area)
InsetRect(TempRect, 3, 3); (Surround the TE area)
if (WidthFieldhdl <> nil) then
    begin          (Update the TE area)
        TEUpdate(TempRect, WidthFieldhdl);
        (Update the TE area)
        TextFont(applFont);
        (Set the default application font)
    end;          (End of the TE area)
(Update a TE box, Height )
SetRect(TempRect, 326, 82, 376, 102);
FrameRect(TempRect); (Frame this TE area)
InsetRect(TempRect, 3, 3); (Surround the TE area)
if (HeightFieldhdl <> nil) then
    begin          (Update the TE area)
        TEUpdate(TempRect, HeightFieldhdl);
        (Update the TE area)
        TextFont(systemFont);
        (Set the default application font)
    end;          (End of the TE area)
(Update a List, Titles )
SetRect(TempRect, 21, 21, 204, 117);
TempRect.Right := TempRect.Right - 15;
(Go inside the scroll bar)
InsetRect(TempRect, -1, -1); (Surround the List area)
FrameRect(TempRect); (Frame this List area)
if (List_I_Titles <> nil) then
    LUpdate(MyWindow.visRgn, List_I_Titles);
    DrawControls(MyWindow); (Draw all the controls)
    SetPort(SavePort); (Restore the old port)
end;          (End for if (MyWindow<>nil))
end;          (End of procedure)

(=====)
(Open our window and draw everything)
procedure Open_Options;
var

```

```

Index: Integer;      (For looping)
dataBounds: Rect;    (For making lists)
cSize: Point;        (For making lists)
(This is a routine used to add strings to an existing list)
procedure Add_List_String (theString: Str255; theList:
ListHandle);
var
    theRow: integer;    (The Row that we are adding)
begin
    if (theList <> nil) then
        begin
            cSize.h := 0;    (Point to the correct column)
            theRow := LAddRow(1, 200, theList);
            cSize.v := theRow;    (Point to the row just added)
            sTemp := theString;    (Get the string to add)
            LSetCell(Pointer(Ord4(0sTemp) + 1), length(sTemp),
cSize, theList); (Place string in row just created)
            LDraw(cSize, theList); (Draw the new string)
        end;
    end;
begin                (Start of Window open routine)
    if (MyWindow = nil) then
        begin
            MyWindow := GetNewWindow(1, nil, Pointer(-1));
            SetPort(MyWindow);    (Prepare to write into window)
            (Open a TE box, Width )
            SetRect(TempRect, 220, 82, 270, 102);
            FrameRect(TempRect); (Frame this TE area)
            InsetRect(TempRect, 3, 3); (Restore the original size)
            WidthFieldhnd1 := TNew(TempRect, TempRect);
            if (theInput <> nil) then
                TEDeactivate(theInput); (Yes, so turn it off)
                theInput := WidthFieldhnd1; (Activate the TE area)
                HLock(Handle(WidthFieldhnd1));
                WidthFieldhnd1^.txFont := systemFont;
                WidthFieldhnd1^.fontAscent := 12; (Font ascent)
                WidthFieldhnd1^.lineHeight := 12 + 3 + 1;
                HUnlock(Handle(WidthFieldhnd1));
                sTemp := '8.50';
                TSetText(Pointer(Ord4(0sTemp) + 1), length(sTemp),
WidthFieldhnd1); (Place default text in the TE area)
                TEActivate(WidthFieldhnd1);
                TEUpdate(TempRect, WidthFieldhnd1);
                TextFont(applFont);
            (Open a TE box, Height )
            SetRect(TempRect, 326, 82, 376, 102);
            FrameRect(TempRect); (Frame this TE area)
            InsetRect(TempRect, 3, 3); (Restore the original size)
            HeightFieldhnd1 := TNew(TempRect, TempRect);
            if (theInput <> nil) then
                TEDeactivate(theInput); (Yes, so turn it off)
                theInput := HeightFieldhnd1; (Activate the TE area)
                HLock(Handle(HeightFieldhnd1));
                HeightFieldhnd1^.txFont := systemFont;
                HeightFieldhnd1^.fontAscent := 12; (Font ascent)
                HeightFieldhnd1^.lineHeight := 12 + 3 + 1;
                HUnlock(Handle(HeightFieldhnd1));
                sTemp := '11.0';
                TSetText(Pointer(Ord4(0sTemp) + 1), length(sTemp),
HeightFieldhnd1); (Place default text in the TE area)
                TEActivate(HeightFieldhnd1); (Make the TE area active)
                TEUpdate(TempRect, HeightFieldhnd1);
                TextFont(applFont);
            (Open a TE box, Title )
            SetRect(TempRect, 220, 40, 380, 62);
            FrameRect(TempRect); (Frame this TE area)
            InsetRect(TempRect, 3, 3); (Restore the original size)

```

```

TitleFieldhnd1 := TNew(TempRect, TempRect);
if (theInput <> nil) then
    TEDeactivate(theInput); (Yes, so turn it off)
    theInput := TitleFieldhnd1; (Activate the TE area)
    HLock(Handle(TitleFieldhnd1));
    TitleFieldhnd1^.txFont := systemFont;
    TitleFieldhnd1^.fontAscent := 12; (Font ascent)
    TitleFieldhnd1^.lineHeight := 12 + 3 + 1;
    HUnlock(Handle(TitleFieldhnd1));
    sTemp := 'US Letter';
    TSetText(Pointer(Ord4(0sTemp) + 1), length(sTemp),
TitleFieldhnd1); (Place default text in the TE area)
    TEActivate(TitleFieldhnd1); (Make the TE area active)
    TEUpdate(TempRect, TitleFieldhnd1);
    TextFont(applFont);
    (Make a List, Titles )
    SetRect(TempRect, 21, 21, 204,
117); (left,top,right,bottom)
    Rect_ListTitles := TempRect; (Save the list position)
    TempRect.Right := TempRect.Right - 15; (Go inside the
scroll bar area)
    InsetRect(TempRect, -1, -1); (Surround the List area)
    FrameRect(TempRect); (Frame this List area)
    InsetRect(TempRect, 1, 1); (Restore to the List area)
    SetRect(dataBounds, 0, 0, 1, 0); (Make the empty list)
    cSize.h := TempRect.Right - TempRect.Left;
    (Get the width of the list)
    cSize.v := 16; (Set the HEIGHT of each list element)
    List_ListTitles := LNew(TempRect, dataBounds, cSize, 0,
MyWindow, TRUE, FALSE, FALSE, TRUE); (Create the list)
    List_ListTitles^.selFlags := 1onlyOne +
1NoNilHilite; (Set for only one active item at a time)
    LdoDraw(TRUE, List_ListTitles); (Draw list)
    cSize.h := 0; (Point to correct col, starts at zero)
    Add_List_String('US Letter', List_ListTitles);
    Add_List_String('A4 Letter', List_ListTitles);
    Add_List_String('US Legal', List_ListTitles);
    Add_List_String('International Fanfold',
List_ListTitles);
    Add_List_String('Computer Paper', List_ListTitles);
    Add_List_String('?', List_ListTitles);
    cSize.h := 0;    (All elements are in column 0)
    cSize.v := 0;    (Select the first list element)
    LSetSelect(TRUE, cSize, List_ListTitles);
    ShowWindow(MyWindow); (Show the window now)
    SelectWindow(MyWindow); (Bring our window to the
front)

    SetDefaultPrec;
end                (End for if (MyWindow<>nil))
else
    SelectWindow(MyWindow); (Already open, so show it)
end;                (End of procedure)

(=====)
(Handle action to our window, like controls)
procedure Do_Options;
var
    RefCon: longint; (RefCon for controls)
    code: integer; (Location of event in window or controls)
    theValue: integer; (Current value of a control)
    whichWindow: WindowPtr;
    myPt: Point; (Point where event happened)
    theControl: ControlHandle; (Handle for a control)
    MyErr: OSErr; (OS error returned)
    DoubleClick: boolean; (Used by lists)
    datalen: integer;
begin                (Start of Window handler)

```

```

Recordstatus;
if (MyWindow <> nil) then
  begin
    code := FindWindow(myEvent.where, whichWindow);
    if (myEvent.what =MouseDown) and (MyWindow =
whichWindow) then()
      begin
        myPt := myEvent.where; (Get mouse position)
        with MyWindow.portBits.bounds do
          begin
            myPt.h := myPt.h + left;
            myPt.v := myPt.v + top;
          end;
        SetRect(tempRect, 220, 40, 380, 62);
        if PtInRect(myPt, tempRect) then
          begin
            if (theInput <> nil) then
              TEDeactivate(theInput);
            theInput := TitleFieldhnd1;
            TEActivate(theInput); (Turn it on)
            TEClick(myPt, FALSE, TitleFieldhnd1);
          end;
        SetRect(tempRect, 220, 82, 270, 102);
        if PtInRect(myPt, tempRect) then
          begin
            if (theInput <> nil) then
              TEDeactivate(theInput);
            theInput := WidthFieldhnd1;
            TEActivate(theInput); (Turn it on)
            TEClick(myPt, FALSE, WidthFieldhnd1);
          end;
        SetRect(tempRect, 326, 82, 376, 102);
        if PtInRect(myPt, tempRect) then
          begin
            if (theInput <> nil) then
              TEDeactivate(theInput);
            theInput := HeightFieldhnd1;
            TEActivate(theInput); (Turn it on)
            TEClick(myPt, FALSE, HeightFieldhnd1);
          end;
        if PtInRect(myPt, Rect_I_Titles) then
          begin
            DoubleClick := LClick(myPt,
myEvent.modifiers, List_I_Titles);
            if DoubleClick then
              begin
                end;
              begin
                cSize := LLastClick(List_I_Titles);
                datalen := 100;
                LGetCell(Pointer(ord(@sTemp) + 1),
datalen, cSize, List_I_Titles); (Get string in row )
                if not (LGetSelect(false, cSize,
List_I_Titles)) then
                  begin
                    theCell.v := 0;
                    theCell.h := 0;
                    if LGetSelect(true, theCell,
List_I_Titles) then
                      begin
                        cSize := theCell;
                        datalen := 100;
                        LGetCell(Pointer(ord(@sTemp) +
1), datalen, cSize, List_I_Titles); (Get string in row )
                      end;
                    end;
                  end;
                end;
              end;
            end;
          end;
        end;
      end;
    end;
  end;
end;

```

```

TESetText(Pointer(Ord4(0PREC4.TitleLcSize.v + 1)) + 1),
datalen, TitleFieldhnd1);
SetRect(TempRect, 220, 40, 380, 62);
TEUpdate(TempRect, TitleFieldhnd1);
SetTeWidth(cSize.v + 1);
SetTeHeight(cSize.v + 1);
end;
end;
end;

if (MyWindow = whichWindow) and (code = inContent)
then(for our window)
  begin
    code := FindControl(myPt, whichWindow,
theControl); (Get type of control)
    if (code <> 0) then(Check type of control)
      code := TrackControl(theControl, myPt,
nil); (Track the control)
    end; (End for if (MyWindow=whichWindow))
  end; (End for if (MyWindow<>nil))
end; (End of procedure)

(=====)
end. (End of unit)

unit HandleTheMenus;
(File name : HandleTheMenus.Pp )
(Function: Handle all menu selections.)
( This procedure is called when a menu item is selected.)
( There is one CASE statement for all Lists. There is)
( another CASE for all the commands in each List.)
(History: 6/10/89 Original by Prototyper. )
( Created for MacTutor )
( ©1989 Copyright MacTutor All Rights Reserved )
interface
uses
  Globals, About, Options, InitTheMenus;
procedure Handle_My_Menu (var doneFlag: boolean; theMenu,
theItem: integer; var theInput: TEHandle);
implementation
procedure Handle_My_Menu;
const
  L_Apple = 1001; (Menu list)
  C_About_Page_Sizes = 1;
  L_File = 1002; (Menu list)
  C_Open = 1;
  C_Save_As = 2;
  C_Set_Page_Setup_Default = 4;
  C_Quit = 6;
  L_Edit = 1003; (Menu list)
  C_Undo = 1;
  C_Cut = 3;
  C_Copy = 4;
  C_Paste = 5;
var
  DNA, Err: integer; (For opening DAs)
  BoolHolder: boolean; (For SystemEdit result)
  DAName: Str255; (For getting DA name)
  SavePort: GrafPtr; (Save current port when opening DAs)
  where: point; (location of SF dialogs)
  prompt, origname, astring: str255;
  typeList: SFTypelist;
  numtypes, count, oldcount: integer;
  Total_Length, RefNum, i, j, k: integer;
  PRECHnd1, SaveHnd1, hnd1: handle;
begin
  (Start of procedure)
  case theMenu of
    (Do selected menu list)

```

```

L_Apple:
begin
case theItem of
C_About_Page_Sizes:
begin
D_About; (Call a dialog)
end;
otherwise (Handle the DAs)
begin (Start of Otherwise)
GetPort(SavePort); (Save the current port)
GetItem(AppleMenu, theItem, DAName);
(Get the name of the DA selected)
DNA := OpenDeskAcc(DAName);
SetPort(SavePort); (Restore to saved port)
end; (End of Otherwise)
end; (End of item case)
end; (End for this list)
L_File:
begin
case theItem of
C_Open:
begin
(Call the SFGetFile OS routine)
where.h := 80;
where.v := 80;
prompt := '';
numTypes := -1;
SFGetFile(where, prompt, nil, numTypes,
typeList, nil, reply);
if reply.good = true then
begin
RefNum := OpenRFParm(reply.fName,
reply.VRefNum, 3);
if (RefNum = -1) or (ResError <> NoErr)
then
SetDefaultPrec
else
begin
Count := CountResources('PREC');
if Count = 0 then
SetdefaultPrec
else
begin
UseResFile(RefNum);
PREC4Hnd1 :=
if (PREC4Hnd1 = nil) or (ResEr-
ror <> NoErr) then
SetdefaultPrec
else
begin
HNoPurge(PREC4Hnd1);
therealprec4hnd1 :=
for k := 1 to 6 do
begin
PREC4.Size[k].h :=
PREC4.Size[k].v :=
end;
count := -1;
for k := 1 to 6 do
begin
count := count + 1;
j :=
ord(therealprec4hnd1^.title[count]) - 1;

```

```

count := count + 1;
oldcount := count;
PREC4.Title[k] := '';
for count := oldcount
to oldcount + j do
begin
PREC4.Title[k] :=
concat(PREC4.Title[k], therealprec4hnd1^.title[count]);
end;
if PREC4.Title[k] = ''
then
PREC4.Title[k] :=
'?';
count := oldcount + j;
end;
SetWTitle(MyWindow,
reply.fName);
end;
CloseResFile(RefNum);
HPurge(PREC4Hnd1);
DisposeHandle(PREC4Hnd1);
SetFields;
end;
end;
end;
C_Save_As:
begin
(Call the SFPutFile OS routine)
where.h := 80;
where.v := 80;
origname := reply.fName;
prompt := 'Save PREC4 resource to file: ';
SFPutFile(where, prompt, origName, nil,
reply);
if reply.good then
begin
Total_length := 32; (first half of PREC
resource)
for i := 1 to 6 do
Total_length := Total_length +
length(PREC4.Title[i]);
SaveHnd1 := NewHandle(Total_length);
(Create new PREC resource)
if SaveHnd1 <> nil then
begin
HNoPurge(SaveHnd1);
therealprec4hnd1 :=
realprec4hnd1(SaveHnd1);
therealprec4hnd1^.numberOfbuttons
:= 0;
for i := 1 to 6 do
if (length(PREC4.Title[i]) <> 0)
or (PREC4.Title[i] = '?') then
therealprec4hnd1^.numberOfbuttons :=
therealprec4hnd1^.numberOfbuttons + 1;
for k := 1 to 6 do
begin
therealprec4hnd1^.Size[k].h :=
PREC4.Size[k].h;
therealprec4hnd1^.Size[k].v :=
PREC4.Size[k].v;
end;
if length(PREC4.Title[1]) <> 0 then
astring := PREC4.Title[1]
else

```



```

        astring := concat(chr(1), '?');
    for j := 2 to 6 do
        begin
            if length(PREC4.Title[j]) <> 0
            then
                astring := concat(astring,
chr(length(PREC4.Title[j]), PREC4.Title[j])
            else
                astring := concat(astring,
chr(1), '?');
            end;
        therealprec4hnd1^.Title[0] :=
chr(length(PREC4.Title[1]));
        for i := 1 to length(astring) do
            begin
                therealprec4hnd1^.Title[i] :=
astring[i];
            end;
        err := SetVol(nil, reply.VRefNum);
        err := Create(reply.fname,
reply.vRefNum, 'PROF', 'TEXT');
        CreateResFile(reply.fname);
        if ResError = dupFNErr then
            begin
                RefNum :=
OpenRfPerm(reply.fname, reply.VRefNum, 3);
                hnd1 := Get1Resource('PREC',
4);

                if Hnd1 <> nil then
                    begin
                        RmveResource(Hnd1);
                        ChangedResource(Hnd1);
                        UpdateResFile(RefNum);
                        DisposHandle(Hnd1);
                    end;
                end;
            begin
                RefNum := OpenRfPerm(reply.fname,
reply.VRefNum, 3);

                end;
            addressource(SaveHnd1, 'PREC', 4,

                writeresource(SaveHnd1);
                closeresfile(RefNum);
                HPurge(SaveHnd1);
                DisposHandle(SaveHnd1);
                SetWTitle(MyWindow, reply.fName);
            end;
        end;
    end;
C_Set_Page_Setup_Default:
begin
    SetDefaultPREC;
end;
C_Quit:
begin
    doneFlag := TRUE;
end;
otherwise
begin(Start of the Otherwise)
end;(End of Otherwise)
end; (End of item case)
end; (End for this list)
L_Edit:
begin
    BoolHolder := SystemEdit(theItem - 1);
    if not (BoolHolder) then

```

```

begin (Handle by using a Case statment)
case theItem of
C_Undo:
begin
end;
C_Cut:
begin
end;
C_Copy:
begin
end;
C_Paste:
begin
end;
otherwise(Send to a DA)
begin(Start of the Otherwise)
end;(End of Otherwise)
end;(End of not BoolHolder)
end; (End of item case)
end; (End for this list)
otherwise
begin (Start of the Otherwise)
end; (End of Otherwise)
end; (End for the Lists)
HiliteMenu(0); (Turn menu selection off)
end; (End of procedure Handle_My_Menu)
end. (End of unit)

```

(The Project should have the following files in it:)

- (µRunTime.lib LSP This is for main Pascal runtime library)
- (Interface.lib LSP This is the Mac trap interfaces)
- (InitTheMenus.p This initializes the Menus.)
- (About Modal Dialog)
- (Options Window)
- (HandleTheMenus Handle the menu selections.)
- (Set RUN OPTIONS to use the resource file PREC4.RSRC)
- (RMaker file to use is PREC4.R)
- (PREC4.p Main program)

program PREC4;

(Program name: PREC4.p)

(Function: This is the main module for this program.)

(History: 6/10/89 Original by Prototyper.)

(Created for MacTutor)

(©1989 CopyRight MacTutor All Rights Reserved)

uses

globals, About, Options, InitTheMenus, HandleTheMenus;

var

(Main variables)

myEvent: EventRecord; (Event record for all events)

doneFlag: boolean; (Exit program flag)

code: integer; (Determine event type)

whichWindow: WindowPtr; (See which window for event)

tempRect, OldRect: Rect; (Rect for dragging)

mResult: longint;

theMenu, theItem: integer; (Menu list and item selected)

chCode: integer; (Key code)

ch: char; (Key pressed in Ascii)

theInput: TEHandle; (Used in text edit selections)

myPt: Point; (Temp Point, used in Zoom)

begin

(Start of main body)

MoreMasters; (This reserves space for more handles)

InitGraf(@thePort); (Quickdraw Init)

InitFonts; (Font manager init)

InitWindows; (Window manager init)

InitMenus; (Menu manager init)

TEInit; (Text edit init)

InitDialogs(nil); (Dialog manager)

```

FlushEvents(everyEvent, 0); (Clear out all events)
InitCursor; (Make an arrow cursor)
doneFlag := FALSE; (Do not exit program yet)
Init_My_Menus; (Initialize menu bar)
theInput := nil; (Init to no text edit selection active)
Init_Options; (Initialize the window routines)
Open_Options(theInput); (Open window routines)
D_About; (Open the modal dialog routines at program start)
reply.fname := '';
repeat (Start of main event loop)
  if (theInput <> nil) then (See if a TE is active)
    TEIdle(theInput);
  SystemTask; (For support of desk accessories)
  if GetNextEvent(everyEvent, myEvent) then
    begin (Start handling the event)
      code := FindWindow(myEvent.where, whichWindow);
      case myEvent.what of (Decide type of event)
        MouseDown: (Mouse button pressed)
          begin (Handle the pressed button)
            if (code = inMenuBar) then
              begin (Get the menu selection and handle it)
                mResult := MenuSelect(myEvent.Where);
                theMenu := HiWord(mResult);
                theItem := LoWord(mResult);
                Handle_My_Menu(doneFlag, theMenu, theItem,
theInput); (Handle the menu)
              end; (End of inMenuBar)
            if (code = InDrag) then
              begin (Do dragging the window)
                tempRect := screenbits.bounds;
                SetRect(tempRect, tempRect.Left + 10,
tempRect.Top + 25, tempRect.Right - 10, tempRect.Bottom - 10);
                DragWindow(whichWindow, myEvent.where,
tempRect); (Drag the window)
              end; (End of InDrag)
            if ((code = inGrow) and (whichWindow <> nil))
then (In a grow area of the window)
              begin (Handle the growing)
                SetPort(whichWindow);
                myPt := myEvent.where; (Get mouse position)
                GlobalToLocal(myPt); (Make it relative)
                OldRect := WhichWindow^.portRect;
                with screenbits.bounds do (screens size)
                  SetRect(tempRect, 15, 15, (right -
left), (bottom - top) - 20); (l,t,r,b)
                mResult := GrowWindow(whichWindow,
myEvent.where, tempRect); (Grow it)
                SizeWindow(whichWindow, LoWord(mResult),
HiWord(mResult), TRUE); (Resize to result)
                SetPort(whichWindow);
                SetRect(tempRect, 0, myPt.v - 15, myPt.h +
15, myPt.v + 15); (Position for horz scrollbar area)
                EraseRect(tempRect); (Erase old area)
                InvalRect(tempRect); (Flag us to update it)
                SetRect(tempRect, myPt.h - 15, 0, myPt.h +
15, myPt.v + 15); (Position for vert scrollbar area)
                EraseRect(tempRect); (Erase old area)
                InvalRect(tempRect); (Flag us to update it)
                DrawGrowIcon(whichWindow);
              end; (End of doing the growing)
            if (code = inZoomIn) or (code = inZoomOut)
then (Handle Zooming windows)
              begin()
                if (WhichWindow <> nil) then
                  begin()
                    SetPort(whichWindow);
                    myPt := myEvent.where;

```

```

GlobalToLocal(myPt); (Make it relative)
OldRect := whichWindow^.portRect;
if TrackBox(whichWindow, myPt, code)
then (Zoom it)
  begin()
    ZoomWindow(WhichWindow, code,
TRUE); (Resize to result)
    SetRect(tempRect, 0, 0, 32000,
32000);
    EraseRect(tempRect);
    InvalRect(tempRect);
  end;
end;
if (code = inGoAway) then
  begin (Handle the goaway button)
    if TrackGoAway(whichWindow, myEvent.where)
then (See if mouse released in GoAway box)
      begin (Handle the GoAway)
        case (GetWRefCon(whichWindow)) of
          1:
            begin
              Close_Options(whichWindow,
theInput); (Close this window)
              doneFlag := TRUE;
            end;
          otherwise (Handle others dialogs)
            begin (Others)
              end; (End of the otherwise)
            end; (End of the case)
          end; (End of TrackGoAway)
        end; (End of InGoAway)
      if (code = inContent) then
        begin (Handle the hit inside a window)
          if (whichWindow <> FrontWindow) then
            SelectWindow(whichWindow)
          else
            begin (Handle the button in the content)
              SetPort(whichWindow);
              case (GetWRefCon(whichWindow)) of
                1:
                  Do_Options(myEvent,
theInput); (Handle this window)
                otherwise (Handle others dialogs)
                  begin (Others)
                    end; (End of the otherwise)
                  end; (End of the case)
                end; (End of else)
              end; (End of inContent)
            if (code = inSysWindow) then
              SystemClick(myEvent, whichWindow);
            end; (End of MouseDown)
          KeyDown, AutoKey: (Handle key inputs)
          begin (Get the key and handle it)
            with myevent do (Check for menu command keys)
              begin()
                chCode := BitAnd(message,
CharCodeMask); (Get character)
                ch := CHR(chCode); (Change to ASCII)
                if (Odd(modifiers div CmdKey)) then
                  begin()
                    mResult := MenuKey(ch);
                    theMenu := HiWord(mResult);
                    theItem := LoWord(mResult);
                    if (theMenu <> 0) then
                      Handle_My_Menu(doneFlag, theMenu,
theItem, theInput); (Do the menu selection)

```

```

        if ((ch = 'x') or (ch = 'X')) and
(theInput <> nil) then()
            TECut(theInput);
        if ((ch = 'c') or (ch = 'C')) and
(theInput <> nil) then()
            TECopy(theInput);
        if ((ch = 'v') or (ch = 'V')) and
(theInput <> nil) then()
            TEPaste(theInput);
        end()
    else if (theInput <> nil) then()
        begin
            if ch = chr(9) then
                begin
                    TEDeactivate(theInput);
                    if theInput = TitleFieldhnd1
                        then
                            theInput := WidthFieldhnd1
                        else if theInput =
WidthFieldhnd1 then
                            theInput := HeightFieldhnd1
                        else if theInput =
HeightFieldhnd1 then
                            theInput := TitleFieldhnd1;
                            TEActivate(theInput); (Turn it on)
                            recordstatus;
                        end
                    else
                        TEKey(ch, theInput);()
                    end;
                end;
            end; (End for KeyDown,AutoKey)
        UpDateEvt: (Update event for a window)
        begin (Handle the update)
            whichWindow := WindowPtr(myEvent.message);
            BeginUpdate(whichWindow);
            case (GetWRefCon(whichWindow)) of
                1:
                    Update_Options(whichWindow);
                otherwise (Handle others dialogs)
                    begin (Others)
                        end; (End of the otherwise)
                    end; (End of the case)
            EndUpdate(whichWindow);
        end; (End of UpDateEvt)
        DiskEvt: (Disk inserted event)
        begin (Handle a disk event)
            if (HiWord(myEvent.message) <> noErr) then
                begin(due to unformatted diskette inserted)
                    myEvent.where.h :=
((screenbits.bounds.Right - screenbits.bounds.Left) div 2) -
(304 div 2); (Center horz)
                    myEvent.where.v :=
((screenbits.bounds.Bottom - screenbits.bounds.Top) div 3) -
(104 div 2); (Top 3ed vertically)
                    InitCursor; (Make sure has arrow cursor)
                    chCode := DIBadMount(myEvent.where,
myEvent.message); (Let the OS handle the diskette)
                end;
            end; (End of DiskEvt)
        app1Evt:
        begin (Start handling our events)
            if (HiWord(myEvent.message) = 1) and
(LoWord(myEvent.Message) = 1) then
                Open_Options(theInput); (Open the window)
            if (HiWord(myEvent.message) = 2) and
(LoWord(myEvent.Message) = 1) then

```

```

            Close_Options(WindowPtr(ord4(-1)),
theInput); (Close the window)
        end; (End handling our events)
        ActivateEvt: (Window activated event)
        begin (Handle the activation)
            whichWindow := WindowPtr(myEvent.message);
            if odd(myEvent.modifiers) then
                begin(Handle the activate)
                    SelectWindow(whichWindow);
                end; (End of Activate)
            end; (End of ActivateEvt)
        otherwise
            end; (End of case)
        end; (end of GetNextEvent)
    until doneFlag; (End of the event loop)
end. (End of the program)

```

```

*****
*
* RMaker resource file sources.
* File: PREC4.R
* History: 6/10/89 Original by Prototyper.
* This file contains the sources for all the resources except
for Pictures.
*
*****

```

```

PREC4.RSRC
APPL???

```

```

*
Type WIND

```

```

        ,1          ;;Resource ID
Page Setup Options          ;;Window title
45 20 185 454              ;;Top Left Bottom Right
InVisible GoAway            ;;Visible GoAway
16                          ;;ProcID, Window def ID
1                          ;;Refcon, reference value

```

```

Type DLOG
*

```

```

        ,2          ;;Resource ID
About                      ;;Dialog title
51 54 182 355              ;;Top Left Bottom Right
Visible NoGoAway            ;;Visible GoAway
1                          ;;ProcID, dialog def ID
2                          ;;Refcon, reference value
2                          ;;ID of item list

```

```

Type DITL
*

```

```

        ,2          ;;Resource ID
3                          ;;Number of controls in list
Button Enabled              ;;Push button
99 109 120 189              ;;Top Left Bottom Right
OK                          ;;message

```

```

StaticText                  ;;Static text
30 80 90 280                ;;Top Left Bottom Right
ImageWriter Page Size Editor\0001989 MacTutor\000By Dave
Kelly;;message

```

```

Icon Enabled                ;;Icon
40 32 72 64                  ;;Top Left Bottom Right

```

31 ;;ID for the resource to use, Icon6

Type ICON = GNRL

* This Icon is Icon6

,32 ;;ICON Resource ID

.H

00000000 001C0000 003E0000 007F8000
00FFE000 01FFF800 03FFFC00 07BFC000
07FEF800 03FFB000 017FF800 021FF400
02E79200 02D11700 02001200 02001700
02809200 02FF9000 0277176C 01002254
00804254 007F8000 00408000 007F8000
01806000 06001800 08618400 130C3200
10000200 1FFFFE00 00000000 00000000

.I

,10032 ;;ICON highlighted Resource ID

.H

00000000 00010000 00038000 0006C000
000D6000 001B8000 0037D800 006FEC00
00DFF600 01BFFB00 037FFD80 06FFFFE0
0DFFFF60 1BFFFFB0 37FFFFD8 6FFFFFEC
F7FFFFDE 73FFFF9C 31FFFF18 18FFFFE3
0C7FFC60 063FF8C0 031FF180 018FE300
00C7C600 00638C00 00311800 00183000

000C6000 0006C000 00038000 00010000

.I

Type MENU

*

,1001 ;;Resource ID
\14 ;;APPLE menu title
About Page Sizes... ;;item title
(- ;;

,1002 ;;Resource ID
File ;;menu title
Open.../O ;;item title
Save As.../S ;;item title
(- ;;
Set Page Setup Default ;;item title
(- ;;
Quit/Q ;;item title

,1003 ;;Resource ID
Edit ;;menu title
(Undo/Z ;;item title
(- ;;
Cut/X ;;item title
Copy/C ;;item title
Paste/V ;;item title



MacApp Workshop

A Tale of Two Quadratic Plotters



Chuck McMath
Carl Nelson
MacApp Developer's Assoc.
Volume 5, Number 8

[Author's Note: This article is NOT about MacApp. We do not cover the basics which you can get elsewhere. For example, this article does not discuss the concepts behind Object programming and Object Pascal syntax, nor do we explain the organization of all the MacApp classes. The December 1987 MacTutor, Kurt Schmucker's excellent book Object Oriented Programming for the Macintosh, the January 1989 APDALog, the newsletter of the MacApp Developer's Association (Frameworks) or the MacApp documentation itself are good reference sources. You may understand some of this article without knowing the basics behind Object Pascal or MacApp, but don't count on it! The programs discussed in this article were written using MacApp 2.0β8. By the time this is published, the final MacApp 2.0 should be available. As far as we know, no changes to this code should be necessary to compile, link, and run it under the final version of MacApp 2.0, but one never knows...]

Introduction

This article is about the conversion of Dave Kelly's and Dave Smith's Plot Program into a MacApp program. The program originally appeared as part of the *Multifinder Friendly MacDraw Plotter* article in the February 1988 MacTutor. Actually, we will discuss two conversions, one written by Chuck McMath (this month) which attempts to literally translate the original program into MacApp and the other a conversion (next month), by Carl Nelson, ports the intent of the original program into a MacApp program. We review the original article and program, then expose the mechanics and decisions involved in doing both conversions. Along the way, we will show you the basics of physically creating a MacApp program and give you some of our reasoning and insight as to why we did what we did. Hopefully you can use this wisdom to do the same as you start writing programs using MacApp. Most of us using MacApp have learned "the hard way" - hours of working on projects until we reach 'enlightenment' on the techniques needed to take advantage of object programming languages.

The motive of this article is not to present the blinding vision of enlightenment on how to learn MacApp but rather a pragmatic approach, an exploration of two ways that might work. We know the starting point (the original program) and the ending point (the original program rewritten using MacApp). Let's travel along the path from start to end with two programs, one a simple clean translation and the other a program that produces the same effect but is internally quite different — the low road (direct translation) and the high road (conversion and expansion). Two points of view, and two results. From both of these you can gain insight into the mechanics of MacApp and a brief look at how Objects and MacApp change the way you look at writing programs.

Introduction Revisited

While Carl's viewpoint is the elegance and power that MacApp gives you in terms of code reusability, my perspective is somewhat different. The main impression that I had when I looked at the 'Multifinder Friendly Plotter' was "hey — this program is all user-interface." Although it does spend some time solving a quadratic equation and fooling around with picture comments, most of the code is involved with the plot parameters dialog box, putting the window up, handling events correctly, etc. Of course, you might say those features are necessary in any Macintosh program, right? Absolutely! These 'common user interface' features are what MacApp is all about. My point of view is that MacApp provides a full-blown user interface, freeing you from having to worry about those details, and letting you concentrate on what makes your application *different*. You can turn your application out more quickly, and have it be of higher quality and greater reliability. The bottom line: less work!

I took the design of the plotter application and decided to implement it just as it was. I didn't make any changes to the overall design to take advantage of MacApp, **but** (and this is an important but) I also did not limit the program to take away any of MacApp's benefits. For instance, the original plotter program puts up a dialog box, waits for you to enter parameters, and then plots the quadratic equation in a new window. If you replot the equation, you get a plot in the same window — *i.e.*, only one window is on the screen at a time. MacApp, however, implements multiple windows as a given; instead of throwing out MacApp's multiple window feature, I allowed the new plotter application to take advantage of that capability. The result is that the new plotter application allows you to produce as many plot windows as memory allows — and better yet — this was done with very little work on my part.

The Original Program

The original plotting program was written just after the new Macintosh ROMs became widely available. Since MacTutor likes to keep on the cutting edge, the two Daves decided to write a program which implements as many features as they could cram into a coherent whole. While the explicit purpose of the program is to solve quadratic equations, the real purpose was to showcase some of the new features in the Mac][/SE ROMs. The new features being exhibited included:

- calling SysEnvirons to determine if the machine supported the required new features
- resources to specify colors in menus/windows
- hierarchical menus used to select colors for parts of the plot
- Multifinder support through calling WaitNextEvent and acting on Multifinder events
- capability to save plots as PICT files, and the use of

PicComments to group logically related items
- hairlines on the LaserWriter through using PicComments.

The operation of the program is fairly simple: when Plot is selected from the File menu, a modal dialog box appears. The user types numbers in for the quadratic equation parameters and some plot parameters, then clicks the OK button (there is no Cancel button). The dialog goes away, and a plot window appears, with the solution to the quadratic equation plotted in the window. The plot is sized so as to always fill the entire window (minus allowance for the scroll bars, of course). A Graph menu lets the user change the color of the axis, the plot, and the background items. A Print Options menu allows the user to specify whether the printed output should be actual size (*i.e.*, the window size) or a full page size. The Graph menu features three hierarchical menus displaying the colors which may be chosen for the three elements of the plot which can be colored (the axes, the plot itself, and the background). The colors available are the 8 supported in the so-called 'Old QuickDraw model.' The window can be resized, dragged, zoomed, and closed. The contents of the window may be saved, and if saved, the file written out is a PICT file which can be opened in MacDraw. If the file is opened in MacDraw, individual items can be manipulated, as if the plot were drawn in MacDraw itself. Selecting Plot again while a plot window is up brings up the modal dialog again, but when it is dismissed, the plot is redrawn in the existing window (only one window is allowed on the screen).

This is only an overview of the original program and this short description cannot do it justice. If you'd like to go read the original article, we won't be offended (just come back to this one.) And now, back to Carl.

MacApp Provides Features

As Chuck said, the original article and program were written to show new features introduced into the Macintosh environment in early 1988 and not as the best or even the right way to do plotting of equations. The original article and the example program show a style for teaching and exploring programming. In order to accomplish their task, the two Daves drew source code from *Professional Programmers Extender*, past issues of *MacTutor*, Dave Wilson's Examples and Steve Sheet's Code, *Inside Macintosh*, and no less than 13 Apple Tech Notes. Several times in the article you can find a Dave saying: "It is much faster to cut and paste source code from someone else, than to reinvent the wheel yourself." The theme of this article, and many MacTutor articles, could just as well have been learning through reuse and extension. Programming by extending and reusing code is one of premises of an application framework and a key feature of languages containing Objects. We show how reuse and extension can occur when bringing the MacTutorPlot program into the MacApp framework. Writing this a year later and having version 2.0B8 of MacApp in hand, many of the concerns presented in the article are non-issues. To be specific:

• Update Events & Growing the Window

MacApp users in general (and specifically for a program the size and complexity of Plot) never worry about update events.

The MacApp architecture nicely handles setting things up for you so that all you have to do is draw when asked to by MacApp. MacApp programmers seldom worry about events, the event loop, Desk Accessories or switching in and out under Multifinder. There is a very robust event handling architecture provided by MacApp and it has been our experience that as the MacOS changes, Apple has enhanced MacApp to match the changes.

• Multifinder

Being Multifinder friendly and having to deal with WaitNextEvent are non-issues. MacApp properly handles all of it as long as the 'SIZE' resource is properly set up. If background processing is needed the right objects should be overridden.

• Palette Manager

The MacApp DrawShapes Example program provides us with an example of how to setup a custom Palette color with a single call to the toolbox GetColor routine.

• Menus (Color, hierarchical or otherwise)

MacApp has a very interesting and useful scheme for handling menus and the actions to be taken when menu selections are made. MacApp uses a resource named 'cmnu' which is the same as a regular 'MENU' but adds an integer to the end. This integer serves as a unique command number for that entry. The cmnu resources are post-processed by a utility called PostRez. It splits apart cmnus and creates a regular MENU and a table that maps these command numbers to menu items for use in your program. MacApp uses these unique command numbers to inform you of menu selections as well as handling the standard kind of menu functions like New, Open and Save. Command numbers 0 through 1200 are reserved for use by MacApp; the rest are available for use in your program. The nice side effect of this is that you can rearrange menus without regard to where they reside in the MENU BAR or their item number. All you ever deal with is the unique command number. Of course there are good examples of custom Menus in both the MacApp Developer's Association 'Goodies Disk' and the MacApp samples shipped by Apple.

• Saving

MacApp provides all the code surrounding New, Save and Save As. If all of your data fits into memory at once, you will have to implement only two procedures, DoNeedDiskSpace and DoWrite, to save your data. You do not have to worry about temp files, disk space and errors, as MacApp provides a powerful error handling mechanism to handle general case failures so you do not have to.

• Printing

By creating a standard print handler object and assigning it to a view (objects which display things in MacApp) a view can be made can be made to print as well as display on the screen. Because of this mechanism, you usually do not have to worry if you are drawing to the screen or a printer port.

• Machine/System Determination

MacApp provides compile time variables (qNeedsScriptManager, qNeedsROM128K, qNeedsHierarchicalMenus, qNeedsStyleTextEdit, qNeedsWaitNextEvent, qNeedsColorQD, qNeedsMC68020, qNeedsMC68030, qNeedsFPU) that you can use to configure your application so that if your

combination of features is not present, your application will safely exit.

With all of the above support, I will not dwell on these features of MacApp [Chuck's note: *I will dwell on them to some extent.*]. We will take this support as given and refer to them as appropriate in our discussions of our conversions. Again, you can find ample documentation as to how these work by reading the MacApp 2.0 final documentation, browsing the sources which are supplied with MacApp or looking through the 1988 and 1989 issues of *Frameworks* which discuss the MacApp 2.0 Architecture.

With all this out of the way we can move on to discuss our conversion efforts. Chuck gets to go first and discuss his efforts then I will talk about my conversion.

Overall Program Design

The crucial point of creating a MacApp program comes before any code has been written: design is the key, for if you have a good object design the code becomes obvious (well, if not 'obvious,' then maybe 'more clear'). A good design can be understood by many people. A bad design is usually not even understood by its creator after a short time passes. The major design of our objects is easy. This is because we are using MacApp. By using MacApp, you 'sign a pact' to become a member of the MacApp community. This agreement says that you will accept all of the wonderful features MacApp provides for your application; in exchange, you agree to be a good citizen and write your application according to a particular structure. This structure enables you to take advantage of the MacApp code, and in a sense, drop your code into MacApp and have it work. The difficult part of object design is deciding where the things you are familiar with fit in. When you start out, you will often be asking yourself which feature and parameters belong to which object. As we'll see, determining this is usually easy, once you understand the objects MacApp provides for us, and how they relate to the pieces of your application. In this particular case, the design is made easier in that we are simply converting an existing program to MacApp, and I am following the original design as much as possible. This means that many of the decisions have been made for us.

MacApp has a number of predefined object classes which equate to the different portions of an application. These predefined objects have the 'generic Mac thing' behavior already implemented. For instance, there is an object which corresponds to the overall application — *TApplication* (incidentally, MacApp object types all begin with a capital T. This is, I suppose, to remind you that an object is simply a declared type in Pascal). An object of type *TApplication* 'knows' how to do all of the things that a good application must — handle desk accessories, put up an about box, work under MultiFinder, launch when double-clicked upon in the Finder, open a document when the document is double-clicked, *etc.* Therefore, you don't have to do anything to your application to have it exhibit this behavior. Instead, you worry about the part of your application that is different from a standard application. One aspect of MacApp programming that you will become accustomed to is that you

don't do all of the work — you let MacApp do it for you. This can make you somewhat confused, for in many cases, you only implement a few methods for an object — yet it seems to be quite a sophisticated object. This is because you are taking advantage of the predefined MacApp object behavior. Don't worry, you will quickly get used to not doing all of the work yourself — and you might even like it.

MacApp	Function	Subclass
<i>TApplication</i>	Manages Documents launch, open, etc.	<i>TQPlotApplication</i>
<i>TDocument</i>	holds data stored on disk	<i>TQPlotDocument</i>
<i>TView</i>	Displays data in windows	<i>TQPlotView</i>

Table 1: MacApp Objects and QPlot Descendents

Table 1 shows a simplified description of the predefined MacApp objects, what they are responsible for, and the descendent objects defined in the plotter application. Keep in mind that when you define an object and modify its behavior, you can be doing one of two things: **adding** some behavior that does not exist, or **changing** some existing behavior. Our *TQPlotApplication* is an example object which was created for both of these reasons. Behavior has to be added to it to handle the colorful hierarchical menus, for example. And some behavior also has to be changed — and this is where experience with MacApp becomes necessary — because the generic MacApp behavior of an application is to open a blank document when you double-click on the application in the Finder. In our case, this behavior is not what we want to happen, so, I decided to override the method which produces the blank document. This requires 'behavior modification' for our *TApplication* object.

As you can see from the table, this application requires only a few simple objects (remember, my goal was to do a quick conversion and not to make the application into the ultimate MacApp program). There is, of course, an application object, since every MacApp program has to have an application object — *TQPlotApplication*. There is a document, since we are concerned with storing data on the disk — *TQPlotDocument*. There is a view, since we are displaying data in a window — *TQPlotView*. And, last, but not least, there is the dialog view, since we have a dialog which requests the plot parameters — *TDialogView*. Note that the dialog view is *not* a specialized view we defined. Why is that? Well, when you look at what a typical Macintosh dialog view lets you do, you will realize that MacApp's dialog functions are ample to do what we want, especially since our dialog is a simple one. The other objects, though, warrant some explanation to show what they provide outside of the generic MacApp structure. Which leads us to the question:

How does anything get done?

Good question, Chuck! Things get done in MacApp by having one of the conceptual 'control chains' take over. The most prevalent 'chain' used in MacApp is the *inheritance chain*. Many parts of MacApp use this sort of chain to pass control on from child (subclass) to parent (superclass). Knowing where to hook in by **OVERRIDEing** is the key to using this and the other chains. The other chains are:

- 1) The *command chain* routes system generated events to objects that want to handle them.
- 2) The *idle chain* is traversed during idle to give selected objects periodic chances to execute.
- 3) The *choice chain* informs a view object's parent views that its state has changed.

Knowing these chains exist help you to determine where as well as who does the work and gives you conceptual frameworks around which to build your application. Now what were you going to say about the plotter application, Chuck?

In the plotter application, nothing happens unless the user selects something using the mouse, so we will only consider mouse down events, which are part of the command chain (incidentally, you don't need to know much about these 'chains' in order to program in MacApp, although knowing what is going on under the hood helps you to understand when things turn out differently from what you expect. But back to mouse downs...) In addition, I'll classify the mouse downs into either menu bar hits or 'other.' Initially I will discuss menu bar hits and how you translate from a menu selection to some activity.

In a non-MacApp program menu enabling and disabling is tied quite closely in with activate and deactivate events. Of course, this makes perfect sense, because when you activate a window you want that window's menu items enabled, and vice versa when the window is being deactivated. There are a few inherent problems with this approach, however. First, you need to have special code in your activate or deactivate event handling procedure to eat up successive activate/deactivate events (the way Chernikoff's MiniEdit does it) or else you will end up in the wrong state — with the wrong items enabled. Second, this technique is not easy to extend or modify, and if you end up adding a new window type, then you add volumes to the code.

MacApp and Menus

In MacApp, menus are handled quite differently, as Carl has described above. MacApp tries to 'unhook' the connection between an item in a menu and an object which handles that item. Instead of hard-coding menu processing in a gigantic procedure, MacApp introduces the concept of commands and command numbers. Think of each menu item as representing a different command, and each command having its unique number. Some commands will best be handled by the application — New, Open, and Quit for example. Other command are the domain of the document — Save, SaveAs and Close. Still other commands should be handled by the object which draws the data — turning grids on and off, aligning objects, perhaps cutting and pasting. So you can see that instead of a central processing repository for all 'menu actions,' MacApp farms the commands out the different

objects. In fact, this is a crucial tenet of object programming: an object will only deal with information it knows how to process. Therefore, in MacApp, each object is given a stab at a menu selection. If the object 'knows' how to handle that command, it does whatever it is supposed to. If, however, the object doesn't recognize that command as something it knows about, the object just passes the request along. Eventually either some object somewhere handled the menu choice, or MacApp senses that no object handled the menu choice, and that choice is simply thrown away.

Menu enabling is handled in the same distributed way as menu selection: there is a special method defined for objects in MacApp where they enable or disable the menu items they are concerned with. This ensures your program will never see a menu item enabled if it shouldn't be, because the only way that item is enabled is if its corresponding object exists. In fact, whenever MacApp determines that the menu bar could have changed, it disables every item in all of the menus. Only after all items have been disabled do the individual existing objects get a chance to enable their functions.

Using a Command to Do Work

When a selection is made from a menu, the object that chooses to handle the menu selection can do one of two things: go ahead and perform whatever action was called for (in easy or trivial cases), or create a command object to perform the command and pass it back to MacApp to be executed (more difficult cases). Command objects are the technique MacApp uses to implement an action and its associated Undo and Redo. Since all of the actions in the plotting program were simple and could be undone easily, my version of the plotter application does not have any command objects. [Carl's Note: My version does]

What about mouse downs that aren't in the menu bar? They are normally handled by a method called DoMouseDownCommand (pretty obvious name, eh? Most MacApp methods attempt to be as obvious) but since the plotter application is a simple one, we don't have anything to do in this method. All of the normal Macintosh behavior is already provided for us by MacApp: dragging the window around; clicking and dragging in the grow box, and resizing the window when we release the button; clicking in the close box to indicate that we want to put the window away; clicking in the zoom box to zoom or unzoom the window — all of these normal, everyday Macintosh features are there — yet we didn't do a thing about them. This is the power of MacApp.

The Application Object

The application object, as we have mentioned before, is responsible for the things a good application does — handling chores at startup and quitting time, creating and opening documents, handling desk accessories, and of course handling and sending events to the appropriate object. What does the object we created for this program do? Remember, that we can either add functionality to our object, or override existing functionality to change the behavior of our customized application object. Outside of initialization code, my application does one thing:

handle choices from the Graph menu, which allows the user to choose the color of the axis, plot, and background. Since we want to be able to set a default for the next plot, the application object has three variables which will correspond to those colors. Selecting from these menus only changes the application's variable. When a plot is created it uses the current set of default colors. Once a plot is created, its colors cannot be changed.

The Document Object

The document object is principally responsible for only one thing: saving and restoring its data. While there are certainly other things the document must be concerned with, most of the document's methods will involve either saving and restoring, or manipulating its data structure — the document is the object which is responsible for maintaining the data that our application requires. In addition, each document will usually have a window associated with it, and it will be used to display data from the document.

The standard document knows that it must be able to save and retrieve itself from disk, however, there's no way the generic document can know how you want to store your data. So, if you were to look in the MacApp source code, in the section that dealt with saving the document, you would find that the DoWrite method is called. However, looking at the DoWrite methods would not help you, since the TDocument.DoWrite method is empty. So what's going on here? Well, this design takes advantage of the *inheritance chain* Carl spoke of earlier. When you define your document class, you will implement your save routine for the DoWrite method that **OVERRIDEs** (replaces) TDocument's DoWrite (this is where you need to understand the concept of **OVERRIDE**). When Save is selected from the File menu, DoWrite is called for the document (your document) and its DoWrite method gets called — all because of the hook put in by MacApp. You don't have to write any code to handle disk errors, not enough space, etc., because MacApp does it for you by calling DoWrite at the right time and under the right conditions. MacApp is full of these hooks — calling a placeholder procedure that serves only to occupy space until you **OVERRIDE** it to provide the needed functionality. In many cases you are only defining these needed **override** procedures. Sometimes it is frustrating and confusing or seems quite magical as to where to find these magic place holders that you can override to get the work done. The only advice we offer here is patience. Reading the documentation sometimes helps, looking at source code of sample programs that behave the way you want is also a good way to discover these placeholders and of course reading articles like this one.

So what does the plotter document do? It has its own initialization code where it clears its copy of the quadratic equation parameters, and creates the window and view. Since the document does this work in my application, it has a method which will present the dialog box to the user and allow the user to type in parameters for the quadratic equation. Solving the equation is another function the document performs (note: the application object could have been chosen to solve the equation; I chose the document object since each document carries its own

set of equation variables). And of course we need methods to write the plot out to disk. Since my plotter application does not read its data back in, there are no methods to read the PICT file, only ones to write it.

The View Object

The view exists to display the data in a window. Since the application draws quadratic equations, this is the main function of the view. However, there are some other functions our view must perform. Since good object design says data should be encapsulated, the view is the only object which knows its size, it is responsible for calculating how large it is. In the plotter application, the view can be one of two sizes: either the view is the entire size of the window or it is the size of a full page. When the window changes size, MacApp sends a message to the view, telling it to recalculate its size. Our view must know whether it is full page size or not, and depending on that status, must calculate its size. Since the view size can be changed by a menu choice, the view must have the necessary code to handle that menu choice. And of course, there must be code to enable the menu choices that the view can handle.

One big difference between the MacApp and the original version of the plotter program is that the original plotter explicitly passed which device was being drawn on — the screen or the LaserWriter — while the MacApp program doesn't know. The original program did this to optimize operations for the LaserWriter (as all good Macintosh programmers do): for instance, when drawing on the LaserWriter, the original program did not fill rectangles with white (since on the LaserWriter this operation is just a waste of time). My MacApp version did not follow this approach because in the MacApp structure, printing is just the same as drawing on the screen — the view doesn't know and shouldn't care which is being done. Although, I could have queried the global variable gPrinting to find out if the view was printing. MacApp takes care of all the device-specific setup, and our view just draws.

The Dialog Object

The dialog which initially appears is a simple one, but it serves to illustrate just how much MacApp does for us, and how much we don't have to worry about. Think about all the steps required to put up a good-looking dialog that simply requests some values: after the resources have been created we need to get the dialog (it's invisible), install a procedure for the UserItem (to outline the default button), stuff initial values, make the dialog visible, drop into a ModalDialog loop, if the default button was pressed then grab values from the dialog items and convert them back into numbers. The design of dialogs in MacApp incorporates much of this tedious process into a complete package that again gets done for us. Sure, MacApp dialogs aren't free, but as they get more complex, MacApp's contribution becomes more and more significant.

Just what is a MacApp dialog?

While in the 'normal' Macintosh environment a dialog is a

special item, in MacApp, a dialog is simply a view (*i.e.*, a place where data gets drawn) with a bunch of views within it. Every item in the DITL that can handle some interaction is a view, and these view types correspond pretty intuitively to the DITL types: TCheckBox, TRadioButton, TButton, TEditText, and TNumberText are the major types. As you may by now assume, these classes respond to mouse clicks in the way we'd want them to: for instance, when the check box is clicked on, the TCheckBox class flips an internal variable and checks the control. So we don't have to do anything special in this dialog to get the kind of interaction we want (in fact, MacApp also does the little things like highlighting the default button, so we have even less work to do).

Our dialog box is pretty simple, since the only items in it that can respond to events are the six parameter EditText items. Each of these items is represented as a TEditText object in MacApp. By now you should realize that each of these 'dialog object types' has default behavior that corresponds to what we expect when we are in a dialog — in our case, the TEditText objects respond to clicks and keystrokes. Hitting a tab while in a field causes the succeeding field (in DITL order) to be highlighted. This behavior is what we want, for the most part.

One important sidelight is that MacApp 2.0 does **not** use the Dialog Manager for its dialogs. This has a number of implications: dialogs are handled like other windows — in fact, you may have noticed that when a modal dialog comes up, the menus all get disabled except for the Edit menu. With MacApp, you can use the Edit menu while your modal dialog is up. Another interesting feature is that since these modal dialogs are not real dialogs, your application can be switched out under MultiFinder while a modal dialog is on top. But back to our description...

You may have noted that there is a TNumberText object, yet we used TEditText objects for numeric fields. Why? Well, TNumberText is only used for LongInts, and we want real numbers, so we will just take the strings and convert them to real numbers ourselves (Unlike Carl, I wasn't aware of Calvin Cock's TExtendedText unit that handles Reals, which he discusses later, so I (ugh) reinvented the wheel. Talk about an advertisement for being in touch.).

What goes where?

One of the decisions you must make when writing a MacApp program is which object a method belongs to. In many cases, it's just not clear which object should have the responsibility. Take cut and paste for example. On the one hand, you could consider that cut and paste change the data structure associated with the document, so they should be document-level methods. On the other hand, you could say that cutting and pasting change the appearance of the items on the screen, so they should be view-level methods. One way to tell that a method is in the wrong place is that you are constantly (*i.e.*, more than two or three times) having to refer to another object's fields to accomplish even a simple task. You probably have the method in the wrong class. Putting a method in the wrong class makes the method look messier or more complicated than it is. You will usually find yourself doing something like:

```
a:= TMyDoc.fMyView.fMyViewData.fGetValue;
```

Another way to tell if a method is "misassigned" is to see if you rely on information that isn't plainly available to your object to enable or disable the menu item associated with that command. If, for instance, you have cut and paste associated with the document, and you need to ask the view if any items are highlighted so you can tell if you should enable the menu items, then you either have a bad design or you have the method in the wrong class.

Discussion of the program

At this point, I'd like to move through the source files and point out where you will find the pieces I have described. The source files consist of three types of files: Pascal (MacApp) source, resources, and the makefile. The source files will merit most of the discussion, so let's start with them.

Main program file (MOPlot.p)

All MacApp main programs look pretty much alike. After the necessary junk at the beginning of the file, the main program starts. Main programs are boring, because so much of the Macintosh functionality is rolled up into the application object. A MacApp main program does the following: initialize the Toolbox Managers and other needed functions, create the application object, call its initializing method, then call its Run method. That's it, and that's usually always it; when Run returns the program execution ends.

Unit file (UOPlot.p and UOPlot.inc1.p)

MacApp source files have a strange arrangement forced upon them by MacApp. The .inc1 source file is organized by object hierarchy: first the application, then the document, and last the view (remember, the dialog view does not have a customized object because the standard dialog handling was sufficient for our purposes).

As we mentioned before, the application object is a simple one, since there is not too much to do. In order to handle the menu choices relating to the color selections, we need to override DoSetupMenus and DoMenuCommand, and both of those methods are pretty easy to understand. Remember, the DoSetupMenus method is called by MacApp frequently (you won't believe just how frequently) so there shouldn't be any lengthy calculations here. If you need to do some complicated calculations to determine if some menu item should be enabled, you should do it somewhere else (like in the *idle chain*) and set a flag or value in your object; that flag can then be used for the menu enabling. The only other method of interest is a real short one: HandleFinderRequest. MacApp's default behavior for applications is that when an application is launched from the Finder by double clicking on the application icon, a blank document opens. Well, we don't want that to happen, since our document is the plot, and we wouldn't want a blank plot window on the screen. So, to modify this behavior, we just override HandleFinderRequest and do nothing. Now when the application is launched, our HandleFinderRequest is called instead of the default one, and we don't get a empty document. This method is a demonstration of

the small things you sometimes have to do to make your application 'perfect' — and note that it's not difficult, it just requires knowing what's going on and where to look. *[Carl's Note: In many ways this typifies the MacApp design - the methods are there to do the kind of things you will need to do - you just have to look for them.]*

The document methods can be characterized into three categories: initialization, solving the quadratic equation, and saving the PICT data. Initialization methods are not very interesting, except to note that the `DoMakeViews` and `DoMakeWindows` methods are some more of the methods which you are required to **OVERRIDE** or else your program will not work (Actually that's not really the truth — with MacApp 2.0 you can omit these overrides and your program will work, it's just that you'll get the default view and window size/location). The method titles are fairly self explanatory. Solving the quadratic equation involves a few methods which contain a lot of code lifted directly from the original plotter application. All I did for these methods was put the method syntax before and after them, and change the variable references to refer to the document object's variables instead of some global variables. The last few methods deal with saving the data to disk. As mentioned before, there are really two methods necessary to save data on disk: `DoNeedDiskSpace` — which calculates if the file will fit on the disk, and `DoWrite` — which actually writes out the data. If there seems to be a lot missing from the implementation of the disk handling, remember that MacApp does most of the work for us — opening the file and checking for errors. Again, all we have to do is handle our application's specific needs, which in this case is saving the PICT.

The view object contains some initialization methods (which should be quite familiar by now). After that is a method called `CalcMinSize`. This method is called (by MacApp of course) when the size of the view needs to be determined. Remember, the view can be either the size of the window or the size of an entire page. So we need to have a flag (`fOnePage`) which tells us which is the case. If we are an entire page, we tell MacApp that our size is the size of our print handler's page (since we want the printout to be one page). If we are not the size of a page, we tell MacApp our size is the size of our superview (our superview is the view which we are inside. In the case of a simple view, our superview is the window itself). After the menu handling routines (`DoSetupMenus` and `DoMenuCommand`) we have a large method which prints the plot on the LaserWriter (again, this method was stolen directly from the original plotter application and had some method syntax wrapped around it). The next two methods were written to handle the case where the window is resized. In `SuperViewChangedSize`, we check to see if we are displaying the plot in full page mode. If not, when the window changes size we need to change the size of the view (since the view mirrors the window size). The next method, `Resize`, has been overridden because when the view size changes we want to force the view to be redrawn (i.e., when switching between full page and window size display modes). The last view method just draws the view, and it is also pretty simple.

Resource file (OPlot.r)

Resources for a MacApp program are almost identical to those for a non-MacApp program. There are only two differences (and they are big differences): 'cmnu' instead of 'MENU' and 'view' resources. 'cmnu' resources (probably stands for 'Command Menu') are used to associate menu items with command numbers. Carl and I have both spoken of the usefulness of command numbers, and I don't need to reiterate it here. The view resource is different, and is a crucial resource in your MacApp program. Carl will speak about view resources in greater detail, but let me say that the view resource allows you to specify the contents of your windows (in functional areas). You can (and will) use views to specify your dialogs in MacApp, because these dialogs are build out of views, as we mentioned earlier. The benefit of using a view resource is the same as you get anywhere else in using resources: you can change the resource without having to recompile the source code. While it may seem that this benefit is limited with regard to views, I'm sure you can attest to the value of resources in other areas, and isn't it nice to be able to 'tweak' a resource quickly?

One other thing you should know about MacApp resource files is that MacApp has a default number for the about box. If you can live with a boring about box (i.e., an `ALRT`) then you should create it with id 201. That's because as part of the generic Macintosh application that MacApp implements, when you select the About Application... from the Apple menu, you get `ALRT 201` displayed. If you like that, it's great, and as always, if you don't like it, you can easily override it. In fact, I override it in my program because I want to calculate some system parameters. I'm sure that you realize the proper method is in the Application level object (since of course the application handles requests to display the about box).

Makefile (OPlot.MAmake)

The makefile is something many Lightspeed users are unfamiliar with. MacApp 2.0's `MABuild` process eliminates the need for a makefile in simple cases. I have provided a simple make file, it's not much, as you can see from the listing. When your program becomes more complicated the makefile usually grows proportionally in complexity. For small projects made up of two or three files, the example makefiles in the MacApp samples folder can be used as guides (and of course, you can consult the manual for more explanation).

Trouble In Paradise

One last parenthetical note before we move over to Carl's description of his program. When we both did the conversions, we independently had problems printing the plots on the LaserWriter — PostScript errors. I tore my hair out about it for a few weeks, then decided to really dig down and see what was happening. By strategically inserting debugging statements into my code I finally isolated my bug to a `PicComment` call. The `PicComment` in question sets the linewidth, and looks like this:

```
PicComment(SetLineWidth,2,Handle(Width));
```

However, the real call should look like this:

```
PicComment(SetLineWidth,4,Handle(Width));
```

The problem was that the second parameter to the PicComment is supposed to be the size (in bytes) of the data. The original specification was wrong, yet it worked (we assume) because Lightspeed Pascal must be more forgiving than MPW. Not only was this line a bug, but the Daves even featured the bug in their article (top of page 21)!! Let this be a lesson that just because someone gives you code that works you can't assume that it is completely bug-free.

That just about covers my program; Next month, we'll hear about Carl's.

```
(-----)
(----- File MQPlot.p -----)
(-----)
($P)
(MQPlot.p)
(Copyright © 1989 by Charles F. McMath All rights reserved.)
```

```
( This is the main program for the MacApp
plotter application. The original plotter was
written by Dave Smith and Dave Kelly for MacTutor
Magazine. This conversion was done by Chuck
McMath to demonstrate the same program done using
MacApp. It also hopefully demonstrates some of
the strengths of MacApp!
)
```

```
PROGRAM QPlot;
```

```
USES
```

```
( o MacApp )
UMacApp,
( o Building Blocks )
UDialog, UPrinting,
( o Implementation Use )
SANE, ToolUtils, Fonts, Resources, Script,
PickerIntf, Packages,
( o the PlotUNIT )
UQPlot;
```

```
VAR
```

```
gQPlotApplication: TQPlotApplication;
```

```
BEGIN
```

```
InitUMacApp(5);
InitUDialog;
InitPrinting;

New(gQPlotApplication);
gQPlotApplication.
  IQPlotApplication(kFileType);
gQPlotApplication.Run;
```

```
END.
```

```
(-----)
(----- File UQPlot.p -----)
(-----)
(Copyright © 1989 by Charles F. McMath.
All rights reserved.)
```

```
UNIT UQPlot;
```

```
INTERFACE
```

```
USES
```

```
( o MacApp - this includes all of the
things necessary from the MacApp Library )
UMacApp,
( o Building Blocks )
UDialog, UPrinting,
```

```
( o Implementation Use )
SANE, ToolUtils, Fonts, Resources, Script,
PickerIntf, Packages;
```

```
CONST
```

```
kSignature = 'FGT3'; (App1. signature)
kFileType = 'PICT'; (our filetype)
kQPlotWindowID = 1001;
```

```
TYPE
```

```
TQPlotApplication = OBJECT(TApplication)
  fDefGraph: INTEGER; ( Graph color index )
  fDefAxis: INTEGER; ( Axis color index )
  fDefBack: INTEGER; ( Back color index )
  fPrintOpt: INTEGER; ( Page/Window size )
```

```
PROCEDURE TQPlotApplication.
```

```
  IQPlotApplication
  (itsMainFileType:OSType);
```

```
FUNCTION TQPlotApplication.DoMakeDocument
  (itsCmdNumber:CmdNumber): TDocument;
  OVERRIDE;
```

```
PROCEDURE TQPlotApplication.
```

```
  HandleFinderRequest; OVERRIDE;
```

```
PROCEDURE TQPlotApplication.DoSetupMenus;
  OVERRIDE;
```

```
FUNCTION TQPlotApplication.DoMenuCommand
  (aCmdNumber: CmdNumber): TCommand;
  OVERRIDE;
```

```
PROCEDURE TQPlotApplication.ShowAboutApp;
  END;
```

```
TQPlotDocument = OBJECT(TDocument)
```

```
  fAParam: Real; ( these parameters )
```

```
  fBParam: Real; ( correspond
                  ( to those )
```

```
  fCParam: Real; ( in the quadratic )
```

```
  fStep: Real; ( equation. )
```

```
  fXScale: INTEGER; ( X axis scale )
```

```
  fYScale: INTEGER; ( Y axis scale )
```

```
  fResult: INTEGER; ( real results? )
```

```
  fRoot1: REAL; ( first root )
```

```
  fRoot2: REAL; ( second root )
```

```
  fPlotView: TQPlotView;
```

```
PROCEDURE TQPlotDocument.IQPlotDocument
```

```
  (itsFileType, itsCreator: OSType;
  usesDataFork, usesRsrcFork: BOOLEAN;
  keepsDataOpen, keepsRsrcOpen:
  BOOLEAN);
```

```
PROCEDURE TQPlotDocument.DoMakeWindows;
  OVERRIDE;
```

```
PROCEDURE TQPlotDocument.DoMakeViews
  (forPrinting: BOOLEAN); OVERRIDE;
```

```
PROCEDURE TQPlotDocument.PosePlotDialog;
```

```
PROCEDURE TQPlotDocument.Quad
  (a, b, c: REAL; VAR x1, x2: REAL;
  VAR result: INTEGER);
```

```
FUNCTION TQPlotDocument.SolveIt
  (a, b, c: REAL; VAR x1, x2: REAL):
  INTEGER;
```

```
PROCEDURE TQPlotDocument.DoNeedDiskSpace
```

```
  (VAR dataForkBytes,
  rsrcForkBytes: LONGINT); OVERRIDE;
```

```
PROCEDURE TQPlotDocument.DoWrite
```

```
  (aRefNum: INTEGER;
  makingCopy: BOOLEAN); OVERRIDE;
```

```
END;
```

```
TQPlotView = OBJECT(TView)
```

```

fPlotDoc: TQPlotDocument; ( link to doc )
fOnePage: BOOLEAN; ( if TRUE, page size )
fDrawing: PicHandle; ( our picture! )

PROCEDURE TQPlotView.IQPlotView
    (theDoc: TQPlotDocument;
     theGrafColor, theAxisColor,
     theBackColor: INTEGER);

PROCEDURE TQPlotView.Free; OVERRIDE;

PROCEDURE TQPlotView.CalcMinSize
    (VAR minSize: VPoint); OVERRIDE;
PROCEDURE TQPlotView.DoSetupMenus;
    OVERRIDE;
FUNCTION TQPlotView.DoMenuCommand
    (aCmdNumber: TCommand); TCommand;
    OVERRIDE;

PROCEDURE TQPlotView.PrQDStuff
    (pRect: rect; QDdevice: integer);
PROCEDURE TQPlotView.SuperViewChangedSize
    (delta: VPoint;
     invalidate: BOOLEAN); OVERRIDE;
PROCEDURE TQPlotView.Resize
    (width, height: VCoordinate;
     invalidate: BOOLEAN); OVERRIDE;
PROCEDURE TQPlotView.Draw(are: Rect); OVERRIDE;
END;

```

IMPLEMENTATION
(\$I UQPlot.incl.p)

END .

```

(-----)
(----- File UQPlot.incl.p -----)
(-----)

```

(Copyright © 1989 by Charles F. McMath.
All rights reserved.)

```

CONST
kPlotWindowID = 1001;
kPlotDLOG      = 2000;
kPlotID        = 'PPRM';

```

kStaggerAmt= 16;

(these are command numbers)

```

cGraphColor      = 1201;
cAxisColor        = 1202;
cBackColor        = 1203;

```

```

cGrafBlack       = 10001;
cGrafWhite       = 10002;
cGrafRed         = 10003;
cGrafGreen       = 10004;
cGrafBlue        = 10005;
cGrafCyan        = 10006;
cGrafMagenta     = 10007;
cGrafYellow      = 10008;

```

```

cAxisBlack       = 11001;
cAxisWhite       = 11002;
cAxisRed         = 11003;
cAxisGreen       = 11004;
cAxisBlue        = 11005;
cAxisCyan        = 11006;
cAxisMagenta     = 11007;
cAxisYellow      = 11008;

```

```

cBackBlack       = 12001;
cBackWhite       = 12002;
cBackRed         = 12003;
cBackGreen       = 12004;
cBackBlue        = 12005;

```

```

cBackCyan        = 12006;
cBackMagenta     = 12007;
cBackYellow      = 12008;

```

```

cPrintWS         = 1301;
cPrintPS         = 1302;

```

```

(*
 * The following constants relate to DLOG
 * and ALRT numbers.
 *)
kAboutDLOG       = 1024; ( our about box )
VAR
gStaggerCount:   INTEGER;
colors:          ARRAY [1..8] OF INTEGER;
crString:        Str255;
(-----)

```

```

FUNCTION Int2Str(theInt: INTEGER): Str255;
(*
 * This utility function takes an integer and
 * converts it into a string.
 *)

```

```

VAR
    templong: LongInt;
    tempStr: Str255;
BEGIN
    templong := theInt;
    NumToStr(templong, tempStr);
    Int2Str := tempStr;
END; ( Int2Str )
(-----)

```

```

FUNCTION Real2String(theReal: REAL;
    numDigits: INTEGER): Str255;

```

```

(*
 * This function takes a real number,
 * and converts it to a string with
 * the specified number of digits PAST the
 * decimal point.
 *)

```

```

VAR
    theForm: DecForm;
    xTemp: Extended;
    tempStr: DecStr;
BEGIN
    theForm.style := FixedDecimal;
    theForm.digits := numDigits;

```

```

    xTemp := theReal;
    Num2Str(theForm, xTemp, tempStr);
    Real2String := tempStr;
END; ( Real2String )
(-----)

```

```

FUNCTION String2Real(theString: Str255): REAL;
(*

```

```

 * This function takes a string form of a real
 * number, and converts it into the real number.
 *)

```

```

VAR
    xTemp: Extended;
BEGIN
    xTemp := Str2Num(theString);
    String2Real := Num2Real(xTemp);
END; ( String2Real )
(-----)

```

```

(----- QPlot Application Methods -----)
(-----)

```

```

PROCEDURE TQPlotApplication.IQPlotApplication
    (itsMainFileType: OSType);

```

```

(*
 * This procedure initializes the application.
 * It sets up all of our global variables and
 * fills the color index array.
 *)

```

VAR

```

aQPlotView: TQPlotView;
BEGIN
  IApplication(itsMainFileType);

  crString[0] := CHR(1);
  crString[1] := CHR(13);

  colors[1] := 33;   { black }
  colors[2] := 30;   { white }
  colors[3] := 205; { red }
  colors[4] := 341; { green }
  colors[5] := 409; { blue }
  colors[6] := 273; { cyan }
  colors[7] := 137; { magenta }
  colors[8] := 69;  { yellow }

  fDefGraph := colors[3];
  fDefAxis := colors[1];
  fDefBack := colors[2];
  fPrintOpt := 1;

  gStaggerCount := 0;
END; ( TQPlotApplication.IQPlotApplication )
(-----)
FUNCTION TQPlotApplication.DoMakeDocument
  (itsCmdNumber: CmdNumber): TDocument;
  OVERRIDE;
(*
  *This function is called whenever we're
  * creating a new document. It creates the
  * document object and returns it (properly
  * initialized, of course).
  *)
VAR
  aQPlotDocument: TQPlotDocument;
BEGIN
  New(aQPlotDocument);
  FailNil(aQPlotDocument);
  aQPlotDocument.IQPlotDocument
    (kFileType, kSignature,
     kUsesDataFork, NOT kUsesRsrcFork,
     NOT kDataOpen, NOT kRsrcOpen);

  aQPlotDocument.fSavePrintInfo := FALSE;
  DoMakeDocument := aQPlotDocument;
END; ( TQPlotApplication.DoMakeDocument )
(-----)
PROCEDURE TQPlotApplication.HandleFinderRequest;
  OVERRIDE;
BEGIN
  ( just override this so we don't put up a
    blank document )
END; ( TQPlotApplication.HandleFinderRequest )
(-----)
PROCEDURE TQPlotApplication.DoSetupMenus;
  OVERRIDE;
(*
  *This procedure enables the appropriate menu
  * items for the application.
  *)
BEGIN
  INHERITED DoSetupMenus;

  Enable(cGraphColor, TRUE);
  Enable(cAxisColor, TRUE);
  Enable(cBackColor, TRUE);
(*
  *Enable all of these, and put a check by
  * the current one.
  *)
  EnableCheck(cGrafBlack, TRUE,
    fDefGraph=colors[1]);
  EnableCheck(cGrafWhite, TRUE,
    fDefGraph=colors[2]);

```

```

  EnableCheck(cGrafRed, TRUE,
    fDefGraph=colors[3]);
  EnableCheck(cGrafGreen, TRUE,
    fDefGraph=colors[4]);
  EnableCheck(cGrafBlue, TRUE,
    fDefGraph=colors[5]);
  EnableCheck(cGrafCyan, TRUE,
    fDefGraph=colors[6]);
  EnableCheck(cGrafMagenta, TRUE,
    fDefGraph=colors[7]);
  EnableCheck(cGrafYellow, TRUE,
    fDefGraph=colors[8]);

```

```

  EnableCheck(cAxisBlack, TRUE,
    fDefAxis=colors[1]);
  EnableCheck(cAxisWhite, TRUE,
    fDefAxis=colors[2]);
  EnableCheck(cAxisRed, TRUE,
    fDefAxis=colors[3]);
  EnableCheck(cAxisGreen, TRUE,
    fDefAxis=colors[4]);
  EnableCheck(cAxisBlue, TRUE,
    fDefAxis=colors[5]);
  EnableCheck(cAxisCyan, TRUE,
    fDefAxis=colors[6]);
  EnableCheck(cAxisMagenta, TRUE,
    fDefAxis=colors[7]);
  EnableCheck(cAxisYellow, TRUE,
    fDefAxis=colors[8]);

```

```

  EnableCheck(cBackBlack, TRUE,
    fDefBack=colors[1]);
  EnableCheck(cBackWhite, TRUE,
    fDefBack=colors[2]);
  EnableCheck(cBackRed, TRUE,
    fDefBack=colors[3]);
  EnableCheck(cBackGreen, TRUE,
    fDefBack=colors[4]);
  EnableCheck(cBackBlue, TRUE,
    fDefBack=colors[5]);
  EnableCheck(cBackCyan, TRUE,
    fDefBack=colors[6]);
  EnableCheck(cBackMagenta, TRUE,
    fDefBack=colors[7]);
  EnableCheck(cBackYellow, TRUE,
    fDefBack=colors[8]);

```

```

END; ( TQPlotApplication.DoSetupMenus )
(-----)

```

```

FUNCTION TQPlotApplication.DoMenuCommand
  (aCmdNumber: CmdNumber): TCommand;
  OVERRIDE;

```

```

(*
  *This function handles menu requests that
  * pertain to the application. It ensures that
  * choices that change colors work correctly.
  *)

```

```

BEGIN

```

```

  DoMenuCommand := gNoChanges;
  CASE aCmdNumber OF
    cAboutApp: ShowAboutApp;
    cGrafBlack,
    cGrafWhite,
    cGrafRed,
    cGrafGreen,
    cGrafBlue,
    cGrafCyan,
    cGrafMagenta,
    cGrafYellow:
      fDefGraph := colors[aCmdNumber -
        cGrafBlack + 1];

```

```

    cAxisBlack,
    cAxisWhite,
    cAxisRed,

```

```

cAxisGreen,
cAxisBlue,
cAxisCyan,
cAxisMagenta,
cAxisYellow:
    fDefAxis := colors[aCmdNumber -
                                cAxisBlack + 1];

cBackBlack,
cBackWhite,
cBackRed,
cBackGreen,
cBackBlue,
cBackCyan,
cBackMagenta,
cBackYellow:
    fDefBack := colors[aCmdNumber -
                                cBackBlack + 1];

cPrintWS,
cPrintPS:
    fPrintOpt := aCmdNumber -
                                cPrintWS + 1;

OTHERWISE
    DoMenuCommand := INHERITED
    DoMenuCommand(aCmdNumber);
END; ( case )
END; ( TQPlotApplication.DoMenuCommand )
(-----)
PROCEDURE DrawBox(wPtr: WindowPtr; itemNum: INTEGER);
(*
 *This procedure is the UserItem proc that
 * gets called for the about box.
 *)
VAR
    ityp:      INTEGER;
    itemHdl:   Handle;
    tRect:     Rect;
BEGIN
    GetDItem(DialogPtr(wPtr), 5, ityp, itemHdl, tRect);
    FrameRect(tRect);
END; ( DrawBox )
(-----)
PROCEDURE TQPlotApplication.ShowAboutApp;
(*
 *Show our about box. This version is almost
 * identical to the original, except for the
 * addition of some stuff about MacApp
 * (Incidentally, this is required by the license
 * agreement for programs developed with MacApp!)
 *)
VAR
    idStrHandle:   StringHandle;
    freeSpace:     Size;
    myHeapSpace:   LongInt;
    tempStr1:      Str255;
    tempStr2:      Str255;
    tempStr3:      Str255;
    diaPtr:        DialogPtr;
    itemHit:        INTEGER;

    ityp:          INTEGER;
    itemHdl:       Handle;
    tRect:         Rect;
BEGIN
    idStrHandle :=
        StringHandle(GetResource(kSignature, 0));
    FailNIL(idStrHandle);

    MoveHHI(Handle(idStrHandle));
    HLock(Handle(idStrHandle));

    freeSpace := FreeMem;
    myHeapSpace := MaxMem(freeSpace);
    NumToString(myHeapSpace, tempStr2);
    tempStr2 := concat('Memory = ', tempStr2);

```

```

tempStr3 := '';
tempStr1 := '';
ParamText(idStrHandle, tempStr1, tempStr2, tempStr3);
diaPtr := GetNewDialog(kAboutDLOG, NIL, Pointer(-1));
FailNIL(diaPtr);
GetDItem(diaPtr, 5, ityp, itemHdl, tRect);
SetDItem(diaPtr, 5, ityp, Handle(@DrawBox), tRect);

InitCursor;
ModalDialog(NIL, itemHit);
DisposDialog(diaPtr);

HUnlock(Handle(idStrHandle));
END; ( TQPlotApplication.ShowAboutApp )
(-----)
(----- QPlot Document Methods -----)
(-----)
PROCEDURE TQPlotDocument.IQPlotDocument
    (itsFileType, itsCreator: OSType;
     usesDataFork, usesRsrcFork: BOOLEAN;
     keepsDataOpen, keepsRsrcOpen:
     BOOLEAN);
(*
 *Initialize the QPlot document. For our
 * document, this means that we will present the
 * dialog box that asks for the quadratic
 * parameters, and we'll solve the equation while
 * we're here.
 *)
VAR
    x1, x2:      REAL;
BEGIN
    IDocument(itsFileType, itsCreator, usesDataFork, us-
    esRsrcFork, keepsDataOpen, keepsRsrcOpen);

    PosePlotDialog;
    fResult := SolveIt(fAParam, fBParam,
                                fCParam, x1, x2);
    IF (fResult < -1) THEN
        BEGIN
            fRoot1 := x1;
            fRoot2 := x2;
        END
    ELSE
        BEGIN
            fRoot1 := -999;
            fRoot2 := -999
        END
    END;
    fPlotView := NIL;
END; ( TQPlotDocument.IQPlotDocument )
(-----)
PROCEDURE TQPlotDocument.DoMakeWindows; OVERRIDE;
(*
 *This procedure is called when we have to
 * make a new document (and its associated
 * window). We just get a simple window and shove
 * it on the screen.
 *)
VAR
    aWindow:   TWindow;
BEGIN
    aWindow := NewSimpleWindow(kPlotWindowID,
        kWantHScrollBar, kWantVScrollBar, SELF, fPlotView);
    aWindow.SimpleStagger(kStaggerAmt,
        kStaggerAmt, gStaggerCount);
END; ( TQPlotDocument.DoMakeWindows )
(-----)
PROCEDURE TQPlotDocument.DoMakeViews
    (forPrinting: BOOLEAN); OVERRIDE;
(*
 *This procedure is called when we are making
 * a new document. It creates all views that are
 * needed. In our case, it's only one. We then
 * initialize the view.

```

```

*)
VAR
    plotApp:    TQPlotApplication;
    aQPlotView: TQPlotView;
    aHandler:    TStdPrintHandler;
    tRect:       Rect;
BEGIN
    plotApp := TQPlotApplication(gApplication);

    NEW(aQPlotView);
    FailNIL(aQPlotView);
    aQPlotView.IQPlotView(SELF,
        plotApp.fDefGraph, plotApp.fDefAxis,
        plotApp.fDefBack);

    fPlotView := aQPlotView;
    (*
    *Now make the view printable by creating a
    * print handler.
    *)
    IF NOT(forPrinting) THEN
        BEGIN
            New(aHandler);
            FailNIL(aHandler);
            aHandler.IStdPrintHandler(SELF,
                aQPlotView, FALSE, TRUE, TRUE);
            ( set up for 1/2" margins )
            SetRect(tRect, 35, 35, -35, -35);
            aHandler.InstallMargins(tRect, FALSE);
        END;
    END; ( TQPlotDocument.DoMakeViews )
    (-----)
PROCEDURE TQPlotDocument.PosePlotDialog;
    (*
    *This procedure puts up the dialog that
    * requests the quadratic parameters.
    *)

    FUNCTION CvtEditReal(aText: TEditText):
        Real;
    VAR
        tempStr: Str255;
    BEGIN
        aText.GetText(tempStr);
        CvtEditReal := String2Real(tempStr);
    END; ( CvtEditReal )

    FUNCTION CvtEditInt(aText: TEditText):
        INTEGER;
    VAR
        tempStr: Str255;
        tempLong: LongInt;
    BEGIN
        aText.GetText(tempStr);
        StringToNum(tempStr, tempLong);
        CvtEditInt := tempLong;
    END; ( CvtEditInt )
VAR
    aWindow: TWindow;
    dismisser: IDType;

    aQPlotView: TQPlotView;
    dView: TDialogView;
    anEditText: TEditText;
    aVal: TEditText;
    bVal: TEditText;
    cVal: TEditText;
    stepVal: TEditText;
    xVal: TEditText;
    yVal: TEditText;

    tempStr: Str255;
BEGIN
    (*

```

```

    *First make our calls (which never get
    * executed) to tell the linker we want to create
    * these types of objects.
    *)
    IF gCreateWithTemplates THEN
        BEGIN
            NEW(aQPlotView);
            NEW(dView);
            NEW(anEditText);
        END;
    (*
    *Get the dialog window and find the dialog subview.
    *)
    aWindow := NewTemplateWindow(kPlotDialog, NIL);
    dView := TDialogView(aWindow.
        FindSubView(kPlotID));
    (*
    *Retrieve references to each of the edit
    * text items. We will need to reference them
    * later to get their values.
    *)
    aVal := TEditText(dView.FindSubView('a '));
    bVal := TEditText(dView.FindSubView('b '));
    cVal := TEditText(dView.FindSubView('c '));
    stepVal := TEditText(dView.FindSubView('step'));
    xVal := TEditText(dView.FindSubView('xsc1'));
    yVal := TEditText(dView.FindSubView('yscl'));

    dView.SelectEditText('a ', kRedraw);

    dismisser := dView.PoseModally;
    (*
    *Now that we have the values, convert them
    * to numbers and plug them into the quadratic
    * equation solver. Note that we don't test to
    * see if the OK button was pressed, because in
    * this dialog, you can't exit except by saying
    * OK.
    *)
    fAParam := CvtEditReal(aVal);
    fBParam := CvtEditReal(bVal);
    fCParam := CvtEditReal(cVal);
    fStep := CvtEditReal(stepVal);
    fXScale := CvtEditInt(xVal);
    fYScale := CvtEditInt(yVal);

    aWindow.Close;
    END; ( TQPlotDocument.PosePlotDialog )
    (-----)
PROCEDURE TQPlotDocument.Quad(a, b, c: REAL;
    VAR x1, x2: REAL;
    VAR result: INTEGER);
    (*
    *This procedure is identical to the one
    * written in 'vanilla' Pascal
    *)
    VAR check: real;

    FUNCTION PositiveCalc
        (a, b, check: real): real;
    BEGIN
        PositiveCalc := (-b + sqrt(check)) /
            (2 * a);
    END; ( PositiveCalc )

    FUNCTION NegativeCalc
        (a, b, check: real): real;
    BEGIN
        NegativeCalc := (-b - sqrt(check)) /
            (2 * a);
    END; ( NegativeCalc )

    BEGIN
        result := 0;

```



```

check := (b * b) - (4 * a * c);
IF result = 0 THEN
  BEGIN
    ( Check if double root exists )
    IF check = 0 THEN
      BEGIN
        result := 2;
        x1 := positivecalc(a, b, check);
        x2 := x1;
      END;
    ( Check if real result )
    IF check > 0 THEN
      BEGIN
        result := 1;
        x1 := positivecalc(a, b, check);
        x2 := negativecalc(a, b, check);
      END;
    ( Check if root is complex )
    IF check < 0 THEN
      BEGIN
        result := 3;
        check := -check;
        x1 := positivecalc(a, b, check);
        x2 := negativecalc(a, b, check);
      END;
    END;
  END; ( TQPlotDocument.Quad )
  (-----)
FUNCTION TQPlotDocument.SolveIt(a, b, c: REAL;
  VAR x1, x2: REAL): INTEGER;
(*
  *This function is identical to the one
  * written for the original program.
  *)
VAR
  result: INTEGER;
BEGIN
  IF (a <> 0) THEN
    quad(a, b, c, x1, x2, result)
  ELSE
    result := -1;
    SolveIt := result;
  END; ( TQPlotDocument.SolveIt )
  (-----)
PROCEDURE TQPlotDocument.DoNeedDiskSpace
  (VAR dataForkBytes,
   rsrcForkBytes: LONGINT); OVERRIDE;
(*
  *This procedure calculates the amount of
  * disk space we need to store our document on
  * disk. In our case, it's just the size of the
  * drawing since no additional information is
  * stored in our file.
  *)
BEGIN
  INHERITED DoNeedDiskSpace(dataForkBytes, rsrcForkBytes);

  dataForkBytes := dataForkBytes +
    512 ( header ) +
    GetHandleSize(Handle(fPlotView.fDrawing));
END; ( TQPlotDocument.DoNeedDiskSpace )
(-----)
PROCEDURE TQPlotDocument.DoWrite(aRefNum: INTEGER;
  makingCopy: BOOLEAN); OVERRIDE;
(*
  *This procedure is called to write the data
  * onto the disk.
  *)
VAR
  count: LongInt;
  buffer: ARRAY [1..256] OF INTEGER;
  pHdl: Pichandle;
  i: INTEGER;
  err: OSErr;

```

```

BEGIN
  INHERITED DoWrite(aRefNum, makingCopy);
  (*
  *Now write the blank 512 byte header needed
  * for PICT files.
  *)
  FOR i:= 1 TO 256 DO
    buffer[i] := 0;

    count := 512;
    err := FSWrite(aRefNum, count, @buffer);
    IF ((err<>noErr) OR (count<>512)) THEN
      SysBeep(10); ( ??? error alert )

    pHdl := fPlotView.fDrawing;
    HLock(Handle(pHdl));
    count := GetHandleSize(Handle(pHdl));
    err := FSWrite(aRefNum, count, Ptr(pHdl));
    HUnlock(Handle(pHdl));
    IF ((err<>noErr) OR
      (count <> GetHandleSize(Handle(pHdl)))) THEN
      SysBeep(10); ( ??? error alert )
  END; ( TQPlotDocument.DoWrite )
  (-----)
  (----- QPlot View Methods -----)
  (-----)
PROCEDURE TQPlotView.IQPlotView(theDoc:
  TQPlotDocument; theGrafColor,
  theAxisColor, theBackColor: INTEGER);
(*
  *This procedure is called to initialize a
  * QPlot view. It sets the view's variables to
  * good initial values.
  *)
VAR
  vOrigin: VPoint;
  vSize: VPoint;
  tempPort: GrafPtr;
BEGIN
  SetVPt(vOrigin, 0, 0);
  SetVPt(vSize, 600, 400);

  IView(theDoc, NIL, vOrigin, vSize, sizeVariable, sizeVariable);
  fPlotDoc := theDoc;

  fDrawing := NIL;
  fOnePage := FALSE;
  END; ( TQPlotView.IQPlotView )
  (-----)
PROCEDURE TQPlotView.Free; OVERRIDE;
(*
  *This procedure is called when the view is
  * finished, and we are cleaning up any space we
  * have allocated. We just dispose of the picture
  * handle.
  *)
BEGIN
  KillPicture(fDrawing);
  INHERITED Free;
END; ( TQPlotView.Free )
(-----)
PROCEDURE TQPlotView.CalcMinSize
  (VAR minSize: VPoint); OVERRIDE;
(*
  *This procedure is called to calculate the
  * minimum size of the view. It is called
  * initially when the view is being created, and
  * later on when we tell the view to recalculate
  * its view size.
  *)
VAR
  vSize: Point;
  tWind: TWindow;

```

```

tRect: Rect;
BEGIN
  INHERITED CalcMinSize(minSize);
  IF fOnePage THEN
    BEGIN
      minSize := fPrintHandler.fViewPerPage;
    END
  ELSE
    BEGIN
      IF (fSuperView <> NIL) THEN
        minSize := fSuperView.fSize
      ELSE
        BEGIN
          vSize := Point(0);
          IF SELF.Focus THEN
            QDToViewPt(vSize, minSize);
          END;
        END;
      END;
    END; { TQPlotView.CalcMinSize }
  (-----)
PROCEDURE TQPlotView.DoSetupMenus; OVERRIDE;
(*
 *This procedure is called to enable menu
 * items that pertain to the view.
 *)
BEGIN
  INHERITED DoSetupMenus;

  EnableCheck(cPrintWS, TRUE, NOT fOnePage);
  EnableCheck(cPrintPS, TRUE, fOnePage);
END; { TQPlotView.DoSetupMenus }
(-----)
FUNCTION TQPlotView.DoMenuCommand
  (aCmdNumber: CmdNumber): TCommand;
  OVERRIDE;
(*
 *This function is called to handle menu
 * commands that pertain to the view.
 *)
VAR
  tempVPt: VPoint;
  tRect: Rect;
BEGIN
  DoMenuCommand := gNoChanges;
  IF (aCmdNumber = cPrintWS) OR
    (aCmdNumber = cPrintPS) THEN
    BEGIN
      fOnePage := (aCmdNumber=cPrintPS);
      AdjustSize;
    END
  ELSE
    DoMenuCommand :=
      INHERITED DoMenuCommand(aCmdNumber);
  END; { TQPlotView.DoMenuCommand }
  (-----)
PROCEDURE TQPlotView.PrQDStuff(pRect: rect;
  QDDevice: integer);
(*
 *This is the main drawing procedure for the
 * view. It is (almost) unchanged from the
 * original version. Since in MacApp you don't
 * know who you're drawing for (Display vs.
 * LaserWriter) most of the display-specific code
 * has been deleted. Drawing still works
 * correctly, since we erase the invalid
 * areas before we draw it.
 *)
CONST
  Display = 1;
  LaserWriter = 2;
  ImageWriter = 3;

  NoJust = 0;
  LeftJust = 1;

```

```

  CenterJust = 2;
  RightJust = 3;
  FullJust = 4;
  LinesInParagraph = 5;
(selected MacDraw comments)
  picDwgBeg = 130;
  picDwgEnd = 131;
  picGrpBeg = 140;
  picGrpEnd = 141;
  TextBegin = 150;
  TextEnd = 151;
  StringBegin = 151;
  StringEnd = 153;
  TextCenter = 154;
(postscript comments)
  SetLineWidth = 182;
  PostScriptBegin = 190;
  TextIsPostscript = 194;
  PostScriptEnd = 191;

```

TYPE

```

  widhd1 = ^widptr;
  widptr = ^widpt;
  widpt = Point;

```

TTxtPicRec = PACKED RECORD

```

  tJus : Byte;
  tFlip : Byte;
  tRot : Integer;
  tLine : Byte;
  tCmnt : Byte;
END;

```

VAR

```

  le, tp, ri, bo : integer;
  str1, str2, str3, str4, str5 : str255;
  str6, str7, str8, str9 : str255;
  hPos, vPos, hor, ver : integer;
  x, y, z1, z2 : real;
  rBox, ClipBox : rect;
  Width : Widhd1;
  leading : integer;
  LineNo : integer;
  ParagraphBegin : Point;
  Indent : integer;
  Paragraph : ARRAY[1..LinesInParagraph]
    OF str255;
  TtxtPicRec : TtxtPicRec;
  TtxtPicPtr : QDPtr;
  TtxtPicHdl : QDHandle;
  TextClipRgn : RgnHandle;
  SaveClip : RgnHandle;
  fInfo : FontInfo;

```

BEGIN

```

  SaveClip := NewRgn;
  GetClip(SaveClip);
  ClipRect(pRect);
  TextClipRgn := NewRgn;

```

```

  penNormal;

```

```

  TextFont(geneva);
  TextSize(10);
  TextFace([]);

```

```

  hor := (pRect.right - pRect.left) DIV 2;
  ver := (pRect.bottom - pRect.top) DIV 2;
  Width := Widhd1(NewHandle(sizeof(widpt)));
  Width^^.h := 10;
  Width^^.v := 1;
  TtxtPicPtr := @TtxtPicRec;
  TtxtPicHdl := @TtxtPicPtr;
  TtxtPicRec.tJus := LeftJust;
  TtxtPicRec.tFlip := 0; (no flip)

```

```

TxtPicRec.tRot := 0; (no rotation)
TxtPicRec.tLine := 2; (1 1/2 spacing)
GetFontInfo(fInfo);
leading := fInfo.descent + fInfo.ascent +
  fInfo.leading;
Indent := 2;
WITH fPlotDoc DO
  BEGIN
    x := -fXScale / 2;
    y := fAParam * x * x + (fBParam * x)
      + fCParam;
    hPos := INTEGER(ROUND(x * hor * 2 / fXScale + hor));
    vPos := INTEGER(ROUND(-y * ver * 2 / fYScale + ver));
    z1 := -fBParam / (2 * fAParam);
    z2 := (4 * fAParam * fCParam -
      (fBParam * fBParam)) / (4 * fAParam);
  END;
  le := 2;
  tp := ver + (ver DIV 3);
  ri := 140;
  IF ri >= (hor + hor DIV 3) THEN
    ri := hor + hor DIV 3;
  bo := ver + ver - 2;
  setRect(rBox, le, tp - 14, ri, bo);
  ParagraphBegin.h := 4;
  ParagraphBegin.v := tp;

(Graph Text)
WITH fPlotDoc DO
  BEGIN
    str1 := Int2Str(-fXScale DIV 2);
    str2 := Int2Str(fYScale DIV 2);
    str3 := Int2Str(fXScale DIV 2);
    str4 := Int2Str(-fYScale DIV 2);

    Paragraph[1] := CONCAT(
      'y=ax^2 + bx + c', crString);
    Paragraph[2] := CONCAT('a=',
      Real2String(fAParam,1),
      ', b=',
      Real2String(fBParam,1),
      ', c=',
      Real2String(fCParam,1),
      crString);
    Paragraph[3] := CONCAT('x1=',
      Real2String(fRoot1,2),
      ', x2=',
      Real2String(fRoot2,2),
      crString);
  END;

CASE fPlotDoc.fResult OF
  1 :
    Paragraph[4] :=
      CONCAT(
        'Two Real Roots, x1, x2',
        crString);
  2 :
    Paragraph[4] :=
      CONCAT('Double Root',
        crString);
  3 :
    Paragraph[4] :=
      CONCAT('Two Complex Roots ',
        crString);
  OTHERWISE
    ;
END;
Paragraph[5] := CONCAT('Slope 0 = (',
  Real2String(z1, 1),
  ', ', Real2String(z2, 1), ')',
  crString);

PenNormal;

```

```

BackColor(TQPlotApplication(gApplication).
  fDefBack);
ForeColor(TQPlotApplication(gApplication).
  fDefAxis);

(Drawing Boundary)
(Begin MacDraw Document)
  PicComment(picDwgBeg, 0, NIL);

  PicComment(picGrpBeg, 0, NIL);
  PicComment(SetLineWidth,
    GetHandleSize(Handle(Width)),
    Handle(Width));

  IF QDdevice = Display THEN
    FillRect(pRect, white);
    FrameRect(pRect);

(Two Axis)
  PicComment(picGrpBeg, 0, NIL);
  moveto(0, ver);
  line(hor + hor, 0);
  moveto(hor, 0);
  line(0, ver + ver);
  PicComment(picGrpEnd, 0, NIL);

  ForeColor(TQPlotApplication(gApplication).
    fDefGraph);

(Plot Itself)
  PicComment(picGrpBeg, 0, NIL);
  moveto(hPos, vPos);
  WITH fPlotDoc DO
    REPEAT
      x := x + fStep;
      y := fAParam * x * x + (fBParam * x) + fCParam;
      hPos := integer(ROUND(x * hor * 2 / fXScale + hor));
      vPos := integer(ROUND(-y * ver * 2 / fYScale + ver));
      WITH pRect DO
        IF (hPos < right) AND
          (hPos > left) AND
          (vPos < bottom) AND
          (vPos > top) THEN
          LineTo(hPos, vPos)
        ELSE
          moveto(hPos, vPos);
        UNTIL x >= fXScale / 2;
      PicComment(picGrpEnd, 0, NIL);

      ForeColor(Colors[1]);

(Axis Text)
    moveto(4, ver + 14);
    DrawString(str1);
    moveto(hor - 40, 14);
    DrawString(str2);
    moveto(hor + hor - 50, ver + 14);
    DrawString(str3);
    moveto(hor - 40, ver + ver - 14);
    DrawString(str4);

(Box )
  PicComment(picGrpBeg, 0, NIL);
  PicComment(picGrpBeg, 0, NIL);

  PicComment(SetLineWidth,
    GetHandleSize(Handle(Width)),
    Handle(Width));

  IF QDdevice = Display THEN
    fillRect(rBox, white);
    frameRect(rBox);
    PicComment(picGrpEnd, 0, NIL); (of box)

```

```

GetClip(TextClipRgn);
ClipBox := rBox;
ClipRect(ClipBox);

(Box Text)
PicComment(TextBegin, sizeof(TTTextPicRec),
  Handle(TxtPicHdl));
FOR LineNo := 1 TO LinesInParagraph DO
  BEGIN
    moveto(ParagraphBegin.h,
      ParagraphBegin.v);
    move(Indent, (LineNo - 1) *
      leading);
    DrawString(Paragraph[LineNo]);
  END;
PicComment(TextEnd, 0, NIL);
PicComment(PicGrpEnd, 0, NIL); (of Box/text)
PicComment(PicGrpEnd, 0, NIL); (of sel. all)
picComment(picDwgEnd, 0, NIL); (of drawing)

SetClip(SaveClip);
disposeHandle(handle(width));
DisposeRgn(TextClipRgn);
DisposeRgn(SaveClip);
END; ( TQPlotView.PrQDStuff )
(-----)
PROCEDURE TQPlotView.SuperViewChangedSize
  (delta: VPoint; invalidate: BOOLEAN);
  OVERRIDE;

(*
 * This procedure changes the view size when
 * the window is resized.
 *)
BEGIN
  (*
   * We only need to go through these
   * shenanigans if we are NOT displaying a page-
   * sized picture. This resizes the view's extent
   * rectangle and invalidates the area, forcing the
   * view to redraw the picture in the new size.
   *)
  IF NOT(fOnePage) THEN
    BEGIN
      AdjustSize;
      ForceRedraw;
    END;
  END; ( TQPlotView.SuperViewChangedSize )
  (-----)
PROCEDURE TQPlotView.Resize (width,
  height: VCoordinate; invalidate:
  BOOLEAN); OVERRIDE;

(*
 * This procedure is called from AdjustSize.
 * It is here because when we toggle between Page
 * Size and Window Size plots, we need to tell the
 * view to redraw itself (because it changed size,
 * even though the window didn't).
 *)
BEGIN
  ForceRedraw;
  INHERITED Resize(width, height, FALSE);
  ForceRedraw;
END; ( TQPlotView.Resize )
(-----)
PROCEDURE TQPlotView.Draw(area: Rect); OVERRIDE;
(*
 * This procedure is called to draw the view
 * when it needs to be drawn. The picture we
 * created before is ALWAYS drawn to take
 * up the entire window.
 *)
VAR
  tRect: Rect;

```

BEGIN

```

  GetQDExtent(tRect); ( always draw pict to
    fill entire extent of the view )
  EraseRect(tRect);
  IF (fDrawing=NIL) THEN
    BEGIN
      fDrawing := OpenPicture(tRect);
      PrQDStuff(tRect, 1);
      ClosePicture;
      PenNormal;
    END;
    DrawPicture(fDrawing, tRect);
  END; ( TQPlotView.Draw )

  (-----)
  (----- File UQPlot.r -----)
  (-----)
  /* Copyright © 1989 by Charles F. McMath.
    All rights reserved. */

  #ifdef Debugging
  include "Debug.rsrc";
  #endif
  include "MacApp.rsrc";
  include "Printing.rsrc";
  include "Dialog.rsrc";

  include "QPlot" 'CODE';

  resource 'BNDL' (128) (
    'FGT3',
    0,
    ( 'ICN#',
      ( 0, 128, 1, 129 ),
      'FREF',
      ( 0, 128, 1, 129 )
    )
  );

  resource 'FREF' (128) (
    'APPL', 0, ""
  );

  resource 'FREF' (129) (
    'PICT', 1, ""
  );

  type 'FGT3' as 'STR ';

  resource 'FGT3' (0) (
    "© by Dave Kelly & Dave Smith \nver"
    "4 JAN 1988;"
    " MacApp version by Chuck McMath"
  );

  resource 'WIND' (1001, purgeable) (
    (45, 15, 445, 615),
    zoomDocProc,
    invisible,
    goAway,
    0x0,
    "<<<>>>"
  );

  resource 'view' (1001, purgeable) (
    (
      root, 'WIND', ( 50, 20 ), ( 260, 430 ),
      sizeVariable, sizeVariable, shown,
      enabled,
      Window ( "TWindow", zoomDocProc,
        goAwayBox, resizable,
        modeless,
        ignoreFirstClick,

```

```

        freeOnClosing,
        disposeOnFree,
        closesDocument,
        openWithDocument,
        dontAdaptToScreen,
        stagger, forceOnScreen,
        dontCenter, 'QPLT', "");
    'WIND', 'SCLR', ( 0, 0 ),
    ( 260-kSBarSizeMinus1,
    430-kSBarSizeMinus1 ),
    sizeRelSuperView,
    sizeRelSuperView,
    shown, enabled,
    Scroller ( "TScroller",
    vertScrollBar,
    horzScrollBar,
    0, 0, 16, 16,
    vertConstrain,
    horzConstrain,
    ( 0, 0, 0, 0 ) );
    'SCLR', IncludeViews ( 1002 )
}
);

resource 'view' (1002, purgeable) {
    (
        root, 'QPLT',
        ( 0, 0 ), ( 134, 414 ),
        sizeFixed, sizeFixed,
        shown, enabled,
        View ( "TQPlotView" )
    )
};

resource 'view' (2000, purgeable) {
    ( /* array viewArray: 15 elements */
        root, noID,
        ( 100, 105 ), ( 150, 300 ),
        sizeVariable, sizeVariable,
        notShown, enabled,
        Window {
            "TWindow",
            documentProc,
            noGoAwayBox,
            notResizable,
            modal,
            ignoreFirstClick,
            freeOnClosing,
            disposeOnFree,
            doesntCloseDocument,
            dontOpenWithDocument,
            dontAdaptToScreen,
            dontStagger,
            dontForceOnScreen,
            center,
            noID,
            "Plot Parameters"
        },
        /* [2] */
        root, 'PPRM',
        ( 0, 0 ), ( 150, 300 ), sizeVariable,
        sizeVariable, shown, disabled,
        DialogView ( "TDialogView", 'ok ',
            noID ),
        /* [3] */
        'PPRM', 'ok ',
        ( 110, 230 ), ( 26, 45 ), sizeFixed,
        sizeFixed, shown, enabled,
        Button (
            "TButton",
            0b0,
            ( 1, 1 ),
            sizeable,
            notDimmed,

```

```

            notHilited,
            dismisses,
            ( 0, 0, 0, 0 ),
            plain,
            0,
            ( 0x0, 0x0, 0x0 ),
            "",
            "OK"
        ),
        /* [4] */
        'PPRM', 'a ',
        ( 30, 15 ), ( 20, 45 ),
        sizeFixed, sizeFixed,
        shown, enabled,
        EditText (
            "TEditText",
            0b1111,
            ( 1, 1 ),
            sizeable,
            notDimmed,
            notHilited,
            doesntDismiss,
            ( 2, 2, 2, 2 ),
            plain,
            0,
            ( 0x0, 0x0, 0x0 ),
            "",
            justLeft,
            "1",
            unlimited,
            0b1111000000000000
            0000000000001000000000
        ),
        /* [5] */
        'PPRM', 'b ',
        ( 30, 100 ), ( 20, 45 ), sizeFixed,
        sizeFixed, shown, enabled,
        EditText (
            "TEditText",
            0b1111,
            ( 1, 1 ),
            sizeable,
            notDimmed,
            notHilited,
            doesntDismiss,
            ( 2, 2, 2, 2 ),
            plain,
            0,
            ( 0x0, 0x0, 0x0 ),
            "",
            justLeft,
            "-1",
            unlimited,
            0b1111000000000000
            0000000000001000000000
        ),
        /* [6] */
        'PPRM', 'c ',
        ( 30, 180 ), ( 20, 45 ), sizeFixed,
        sizeFixed, shown, enabled,
        EditText (
            "TEditText",
            0b1111,
            ( 1, 1 ),
            sizeable,
            notDimmed,
            notHilited,
            doesntDismiss,
            ( 2, 2, 2, 2 ),
            plain,
            0,
            ( 0x0, 0x0, 0x0 ),
            "",
            justLeft,

```

```

        "-6",
        unlimited,
        0b111100000000000000000000
00001000000000
    },
    /* [7] */
    'PPRM', 'step',
    (80, 10), (20, 55), sizeFixed,
    sizeFixed, shown, enabled,
    EditText {
        "TEditText",
        0b1111,
        (1, 1),
        sizeable,
        notDimmed,
        notHilited,
        doesntDismiss,
        (2, 2, 2, 2),
        plain,
        0,
        (0x0, 0x0, 0x0),
        "",
        justLeft,
        ".05",
        unlimited,
        0b111100000000000000000000
000100000000
    },
    /* [8] */
    'PPRM', 'xsc1',
    (80, 100), (20, 50), sizeFixed,
    sizeFixed, shown, enabled,
    EditText {
        "TEditText",
        0b1111,
        (1, 1),
        sizeable,
        notDimmed,
        notHilited,
        doesntDismiss,
        (2, 2, 2, 2),
        plain,
        0,
        (0x0, 0x0, 0x0),
        "",
        justLeft,
        "10",
        unlimited,
        0b111100000000000000000000
00001000000000
    },
    /* [9] */
    'PPRM', 'ysc1',
    (80, 185), (20, 50), sizeFixed,
    sizeFixed, shown, enabled,
    EditText {
        "TEditText",
        0b1111,
        (1, 1),
        sizeable,
        notDimmed,
        notHilited,
        doesntDismiss,
        (2, 2, 2, 2),
        plain,
        0,
        (0x0, 0x0, 0x0),
        "",
        justLeft,
        "20",
        unlimited,
        0b111100000000000000000000
001000000000
    },

```

```

/* [10] */
'PPRM', 'alab',
(10, 35), (16, 20),
sizeFixed, sizeFixed,
shown, disabled,
StaticText {
    "TStaticText",
    0b0,
    (1, 1),
    sizeable,
    notDimmed,
    notHilited,
    doesntDismiss,
    (0, 0, 0, 0),
    plain,
    0,
    (0x0, 0x0, 0x0),
    "",
    justLeft,
    "a"
},
/* [11] */
'PPRM', 'blab',
(10, 120), (16, 20), sizeFixed,
sizeFixed, shown, disabled,
StaticText {
    "TStaticText",
    0b0,
    (1, 1),
    sizeable,
    notDimmed,
    notHilited,
    doesntDismiss,
    (0, 0, 0, 0),
    plain,
    0,
    (0x0, 0x0, 0x0),
    "",
    justLeft,
    "b"
},
/* [12] */
'PPRM', 'clab',
(10, 200), (16, 20), sizeFixed,
sizeFixed, shown, disabled,
StaticText {
    "TStaticText",
    0b0,
    (1, 1),
    sizeable,
    notDimmed,
    notHilited,
    doesntDismiss,
    (0, 0, 0, 0),
    plain,
    0,
    (0x0, 0x0, 0x0),
    "",
    justLeft,
    "c"
},
/* [13] */
'PPRM', 'slab',
(60, 10), (16, 55),
sizeFixed, sizeFixed,
shown, disabled,
StaticText {
    "TStaticText",
    0b0,
    (1, 1),
    sizeable,
    notDimmed,
    notHilited,
    doesntDismiss,

```

```
(0, 0, 0, 0),  
plain,  
0,  
{0x0,0x0,0x0},  
"" ,  
justLeft,  
"step size"  
),  
/* [14] */  
'PPRM', 'xlab',  
(60, 95),(16, 55),  
sizeFixed, sizeFixed,  
shown, disabled,  
StaticText (  
    "TstaticText",  
    0b0,  
    {1, 1},  
    sizeable,  
    notDimmed,  
    notHilited,  
    doesntDismiss,  
    {0, 0, 0, 0}),  
plain,  
0,  
{0x0,0x0,0x0},  
"" ,  
justLeft,  
"x scale"  
),  
/* [15] */  
'PPRM', 'y lab',  
(60, 180),(16, 70), sizeFixed,  
sizeFixed, shown, disabled,  
StaticText (  
    "TstaticText",  
    0b0,  
    {1, 1},  
    sizeable,  
    notDimmed,  
    notHilited,  
    doesntDismiss,  
    {0, 0, 0, 0}),  
plain,  
0,  
{0x0,0x0,0x0},  
"" ,  
justLeft,  
"y scale"  
)  
}  
);
```

```
resource 'SIZE' (-1) {  
dontSaveScreen,  
acceptSuspendResumeEvents,  
enableOptionSwitch,  
canBackground,  
MultiFinderAware,  
backgroundAndForeground,  
dontGetFrontClicks,  
ignoreChildDiedEvents,  
is32BitCompatible,  
reserved,  
reserved,  
reserved,  
reserved,  
reserved,  
reserved,  
reserved,  
if qDebug  
275 * 1024,  
200 * 1024  
else  
(250-32) * 1024,
```

```

(175-32) * 1024
#endif
);

resource 'DLOG' (1024) (
    (100, 100, 318, 416),
    dBoxProc,
    -1,
    noGoAway,
    0x0,
    1024,
    "About Plotter..."
);

resource 'DITL' (1024, purgeable) (
    ( /* array DITLarray: 5 elements */
    /* [1] */
    (112, 235, 141, 284),
    Button (
        enabled,
        "OK"
    ),
    /* [2] */
    (10, 88, 141, 289),
    StaticText (
        disabled,
        "Plot Demo\n\nGraphs "
        "Quadratic Equations\n^0"
        "\n^1\n^2\n^3"
    ),
    /* [3] */
    (10, 10, 96, 81),
    Picture (
        disabled,
        128
    ),
    /* [4] */
    (154, 22, 208, 283),
    StaticText (
        disabled,
        "This application brought"
        " to you courtesy"
        " of MacApp® 2.0."
        " Copyright "1985-1988"
        " Apple Computer, Inc."
    ),
    /* [5] */
    (149, 17, 211, 290),
    UserItem (
        disabled
    )
    )
);

resource 'cmnu' (1) (
    1,
    textMenuProc,
    0x7FFFFFFD,
    enabled,
    apple,
    ( /* array: 2 elements */
    /* [1] */
    "About Plotter...", noIcon, noKey, noMark,
    plain, cAboutApp;
    /* [2] */
    "- ", noIcon, noKey, noMark, plain, nocommand
    )
);

resource 'cmnu' (2) (
    2,
    textMenuProc,
    allEnabled,
    enabled.

```

```

"File",
  (/* array: 8 elements */
  /* [1] */
  "Plot...", noIcon, "P", noMark,
plain, cNew;
  /* [2] */
  "-", noIcon, noKey, noMark, plain,
noCommand;
  /* [3] */
  "Save", noIcon, "S", noMark, plain,
cSave;
  /* [4] */
  "Save As...", noIcon, noKey, noMark,
plain,
    cSaveAs;
  /* [5] */
  "Page Setup...", noIcon, "U", noMark,
plain,
    cPageSetup;
  /* [6] */
  "Print...", noIcon, "O", noMark,
plain, cPrint;
  /* [7] */
  "-", noIcon, noKey, noMark, plain,
noCommand;
  /* [8] */
  "Quit", noIcon, "Q", noMark, plain,
cQuit
  )
);

resource 'cmnu' (3) {
  3,
  textMenuProc,
  allEnabled,
  enabled,
  "Edit",
  (/* array: 6 elements */
  /* [1] */
  "Undo", noIcon, "Z", noMark, plain,
cUndo;
  /* [2] */
  "-", noIcon, noKey, noMark, plain,
noCommand;
  /* [3] */
  "Cut", noIcon, "X", noMark, plain,
cCut;
  /* [4] */
  "Copy", noIcon, "C", noMark, plain,
cCopy;
  /* [5] */
  "Paste", noIcon, "V", noMark,
plain, cPaste;
  /* [6] */
  "Clear", noIcon, noKey, noMark,
plain, cClear
  )
);

resource 'cmnu' (4) {
  4,
  textMenuProc,
  allEnabled,
  enabled,
  "Graph",
  (/* array: 3 elements */
  /* [1] */
  "Graph", noIcon, "\0x1B", "\0x0A",
plain,
    1201;
  /* [2] */
  "Axis", noIcon, "\0x1B", "\0x0B",
plain, 1202;
  /* [3] */

```

```

    "Background", noIcon, "\0x1B",
    "\0x0C", plain,
    1203
  )
  );

resource 'cmnu' (5) {
  5,
  textMenuProc,
  allEnabled,
  enabled,
  "Print Options",
  (/* array: 2 elements */
  /* [1] */
  "Window Size", noIcon, "[", noMark,
plain,
    1301;
  /* [2] */
  "Page Size", noIcon, "]", noMark,
plain, 1302
  )
  );

resource 'cmnu' (10) {
  10,
  textMenuProc,
  allEnabled,
  enabled,
  "Graph Colors",
  (/* array: 8 elements */
  /* [1] */
  "Black", noIcon, noKey, noMark,
plain, 10001;
  /* [2] */
  "White", noIcon, noKey, noMark,
plain, 10002;
  /* [3] */
  "Red", noIcon, noKey, noMark,
plain, 10003;
  /* [4] */
  "Green", noIcon, noKey, noMark,
plain, 10004;
  /* [5] */
  "Blue", noIcon, noKey, noMark,
plain, 10005;
  /* [6] */
  "Cyan", noIcon, noKey, noMark,
plain, 10006;
  /* [7] */
  "Magenta", noIcon, noKey, noMark,
plain,
    10007;
  /* [8] */
  "Yellow", noIcon, noKey, noMark,
plain, 10008
  )
  );

resource 'cmnu' (11) {
  11,
  textMenuProc,
  allEnabled,
  enabled,
  "Axis Colors",
  (/* array: 8 elements */
  /* [1] */
  "Black", noIcon, noKey, noMark,
plain, 11001;
  /* [2] */
  "White", noIcon, noKey, noMark,
plain, 11002;
  /* [3] */
  "Red", noIcon, noKey, noMark,
plain, 11003;

```

```

  /* [4] */
  "Green", noIcon, noKey, noMark,
plain, 11004;
  /* [5] */
  "Blue", noIcon, noKey, noMark,
plain, 11005;
  /* [6] */
  "Cyan", noIcon, noKey, noMark,
plain, 11006;
  /* [7] */
  "Magenta", noIcon, noKey, noMark,
plain,
    11007;
  /* [8] */
  "Yellow", noIcon, noKey, noMark,
plain, 11008
  )
  );

resource 'cmnu' (12) {
  12,
  textMenuProc,
  allEnabled,
  enabled,
  "Background Colors",
  (/* array: 8 elements */
  /* [1] */
  "Black", noIcon, noKey, noMark,
plain, 12001;
  /* [2] */
  "White", noIcon, noKey, noMark,
plain, 12002;
  /* [3] */
  "Red", noIcon, noKey, noMark,
plain, 12003;
  /* [4] */
  "Green", noIcon, noKey, noMark,
plain, 12004;
  /* [5] */
  "Blue", noIcon, noKey, noMark,
plain, 12005;
  /* [6] */
  "Cyan", noIcon, noKey, noMark,
plain, 12006;
  /* [7] */
  "Magenta", noIcon, noKey, noMark,
plain,
    12007;
  /* [8] */
  "Yellow", noIcon, noKey, noMark,
plain, 12008
  )
  );

resource 'cmnu' (128) {
  128,
  textMenuProc,
  allEnabled,
  enabled,
  "Buzzwords",
  (/* array: 1 elements */
  /* [1] */
  "Page Setup Change", noIcon, noKey,
noMark,
    plain, cChangePrinterStyle
  )
  );

resource 'MBAR' (128) {
  (1; 2; 3; 4; 5)
  );

resource 'MBAR' (130) {
  (10; 11; 12)

```



```

);
resource 'mctb' (0) {
    (0, 0,
        ( /* array: 4 elements */
            0, 0, 0,
            65535, 65535, 65535,
            65535, 0, 65535,
            65535, 65535, 65535
        )
    )
};

resource 'mctb' (1) {
    (1, 0,
        /* [1] */
        ( /* array: 4 elements */
            0, 65535, 65535,
            65535, 65535, 65535,
            65535, 0, 0,
            65535, 65535, 65535
        ),
        /* [2] */
        1, 1,
        ( /* array: 4 elements */
            0, 0, 65535,
            0, 0, 65535,
            0, 0, 65535,
            65535, 65535, 65535
        ),
        /* [3] */
        1, 2,
        ( /* array: 4 elements */
            0, 0, 65535,
            0, 0, 65535,
            0, 0, 65535,
            65535, 65535, 65535
        )
    )
};

resource 'mctb' (10) {
    (10, 1,
        ( /* array: 4 elements */
            0, 0, 0,
            0, 0, 0,
            0, 0, 0,
            65535, 65535, 65535
        ),
        /* [2] */
        10, 2,
        ( /* array: 4 elements */
            0, 0, 0,
            0, 0, 0,
            0, 0, 0,
            65535, 65535, 65535
        ),
        /* [3] */
        10, 3,
        ( /* array: 4 elements */
            65535, 0, 0,
            65535, 0, 0,
            65535, 0, 0,
            65535, 65535, 65535
        ),
        /* [4] */
        10, 4,
        ( /* array: 4 elements */
            0, 65535, 0,
            0, 65535, 0,
            0, 65535, 0,
            65535, 65535, 65535
        ),
        /* [5] */
        10, 5,
        ( /* array: 4 elements */
            0, 0, 65535,
            0, 0, 65535,
            0, 0, 65535,
            65535, 65535, 65535
        ),
        /* [6] */
        10, 6,
        ( /* array: 4 elements */
            0, 65535, 65535,
            0, 65535, 65535,
            0, 65535, 65535,
            65535, 65535, 65535
        ),
        /* [7] */
        10, 7,
        ( /* array: 4 elements */
            65535, 0, 65535,
            65535, 0, 65535,
            65535, 0, 65535,
            65535, 65535, 65535
        ),
        /* [8] */
        10, 8,
        ( /* array: 4 elements */
            65535, 65535, 0,
            65535, 65535, 0,
            65535, 65535, 0,
            65535, 65535, 65535
        )
    )
};

resource 'mctb' (11) {
    ( /* array MCTBArray: 8 elements */
        /* [1] */
        11, 1,
        ( /* array: 4 elements */
            0, 0, 0,
            0, 0, 0,
            0, 0, 0,
            65535, 65535, 65535
        ),
        /* [2] */
        11, 2,
        ( /* array: 4 elements */
            0, 0, 0,
            0, 0, 0,
            0, 0, 0,
            65535, 65535, 65535
        ),
        /* [3] */
        11, 3,
        ( /* array: 4 elements */
            65535, 0, 0,
            65535, 0, 0,
            65535, 0, 0,
            65535, 65535, 65535
        ),
        /* [4] */
        11, 4,
        ( /* array: 4 elements */
            0, 65535, 0,
            0, 65535, 0,
            0, 65535, 0,
            65535, 65535, 65535
        ),
        /* [5] */
        11, 5,
        ( /* array: 4 elements */
            0, 0, 65535,
            0, 0, 65535,
            0, 0, 65535,
            65535, 65535, 65535
        ),
        65535, 65535, 65535
    ),
    /* [6] */
    11, 6,
    ( /* array: 4 elements */
        0, 65535, 65535,
        0, 65535, 65535,
        0, 65535, 65535,
        65535, 65535, 65535
    ),
    /* [7] */
    11, 7,
    ( /* array: 4 elements */
        65535, 0, 65535,
        65535, 0, 65535,
        65535, 0, 65535,
        65535, 65535, 65535
    ),
    /* [8] */
    11, 8,
    ( /* array: 4 elements */
        65535, 65535, 0,
        65535, 65535, 0,
        65535, 65535, 0,
        65535, 65535, 65535
    ),
    /* [9] */
    11, 9,
    ( /* array: 4 elements */
        65535, 65535, 0,
        65535, 65535, 0,
        65535, 65535, 0,
        65535, 65535, 65535
    )
);

resource 'mctb' (12) {
    ( /* array MCTBArray: 8 elements */
        /* [1] */
        12, 1,
        ( /* array: 4 elements */
            0, 0, 0,
            0, 0, 0,
            0, 0, 0,
            65535, 65535, 65535
        ),
        /* [2] */
        12, 2,
        ( /* array: 4 elements */
            0, 0, 0,
            0, 0, 0,
            0, 0, 0,
            65535, 65535, 65535
        ),
        /* [3] */
        12, 3,
        ( /* array: 4 elements */
            65535, 0, 0,
            65535, 0, 0,
            65535, 0, 0,
            65535, 65535, 65535
        ),
        /* [4] */
        12, 4,
        ( /* array: 4 elements */
            0, 65535, 0,
            0, 65535, 0,
            0, 65535, 0,
            65535, 65535, 65535
        ),
        /* [5] */
        12, 5,
        ( /* array: 4 elements */
            0, 0, 65535,
            0, 0, 65535,
            0, 0, 65535,
            65535, 65535, 65535
        ),
        /* [6] */
        12, 6,
        ( /* array: 4 elements */
            0, 0, 65535,
            0, 0, 65535,
            0, 0, 65535,
            65535, 65535, 65535
        ),
        /* [7] */
        12, 7,
        ( /* array: 4 elements */
            65535, 0, 65535,
            65535, 0, 65535,
            65535, 0, 65535,
            65535, 65535, 65535
        ),
        /* [8] */
        12, 8,
        ( /* array: 4 elements */
            65535, 65535, 0,
            65535, 65535, 0,
            65535, 65535, 0,
            65535, 65535, 65535
        ),
        /* [9] */
        12, 9,
        ( /* array: 4 elements */
            65535, 65535, 0,
            65535, 65535, 0,
            65535, 65535, 0,
            65535, 65535, 65535
        )
    )
};

```

```

0, 65535, 65535,
0, 65535, 65535,
0, 65535, 65535,
65535, 65535, 65535
),
/* [7] */
12, 7,
( /* array: 4 elements */
65535, 0, 65535,
65535, 0, 65535,
65535, 0, 65535,
65535, 65535, 65535
),
/* [8] */
12, 8,
( /* array: 4 elements */
65535, 65535, 0,
65535, 65535, 0,
65535, 65535, 0,
65535, 65535, 65535
)
)
);

resource 'PICT' (128) (
891,
(195, 254, 281, 325),
$'1101 A000 82A0 008E 0100 0A00 0000 0002"
$'D002 4098 000A 00C3 00F8 00FF 0148 00C3"
$'00FE 00FF 0145 00C3 00FE 00FF 0145 0000"
$'02F7 0002 F700 02F7 0002 F700 02F7 0002"
$'F700 02F7 0002 F700 02F7 0002 F700 02F7"
$'0006 FD00 000E FC00 07FD 0001 1F80 FD00"
$'07FD 0001 7FC0 FD00 07FD 0001 FFF0 FD00"
$'08FE 0002 03FF FCFD 0008 FE00 0207 FFFE"
$'FD00 09FE 0003 1FFF FF80 FE00 09FE 0003"
$'3FFF FFE0 FE00 09FE 0003 7FFF FFF8 FE00"
$'0A02 0000 01FE FF00 FCFE 0008 0200 0003"
$'FDFF FE00 0A02 0000 0FFD FF00 C0FF 000B"
$'0700 001F FFFF 3FFF E0FF 000B 0700 007F"
$'FFFE 1FFF F8FF 000B 0700 00FF FFFE 1FFF"
$'FCFF 000B 0100 01FE FF02 27FF FCFE 000B"
$'0100 01FE FF02 F9FF F8FF 000B 0100 00FE"
$'FF02 FE7F F0FF 000B 0200 003F FFFF 019F"
$'E0FF 000B 0200 001F FFFF 01E7 C0FF 000B"
$'0200 003F FFFF 01F9 80FF 000B 0200 0033"
$'FEFF 01FE 80FF 000A 0200 0060 FFFF 00C0"
$'FF00 0B07 0000 607F FFFF FCC0 FF00 0B07"
$'0000 601F FFFF F870 FF00 0B07 0000 6007"
$'FFFF F0F8 FF00 0B07 0000 6001 FFFF F0F8"
$'FF00 0B07 0000 6000 FFFF F0F8 FF00 0B07"
$'0000 6038 3FFF B050 FF00 0A06 0000 607C"
$'0FFF 30FE 000B 0700 0060 F603 FE30 A8FF"
$'000B 0700 0060 E301 FC30 50FF 000B 0700"
$'0060 C000 7830 20FF 000B 0700 0060 0000"
$'1030 88FF 000B 0200 0060 FE00 0130 50FF"
$'000A 0200 0060 FE00 0030 FE00 0B02 0000"
$'60FE 0001 30A8 FF00 0B07 0000 6807 0700"
$'B050 FF00 0A06 0000 681F 8FC0 B0FE 000B"
$'0700 006C 7FDF F1B0 A8FF 000A 0200 0067"
$'FEFF 0030 FE00 0B09 0000 63FF FFFE 31F4"
$'1000 0B09 0000 307F DFF0 6046 3000 0B09"
$'0000 381F 8FC0 E045 5000 0B09 0000 1C00"
$'0001 C044 9000 0B09 0000 0E00 0003 8044"
$'1000 0802 0000 07FE FFFD 0009 0500 0001"
$'FFFF FCFD 0008 FE00 0280 0004 F000 9800"
$'0A00 FF00 F801 1901 4800 FF00 FE01 1901"
$'4500 FF00 FE01 1901 4500 0008 FE00 0280"
$'0004 FD00 08FE 0002 FFFF FCFD 0008 0200"
$'0001 FEAA FD00 0802 0000 03FE 55FD 000A"
$'0600 0006 FEAF EA80 FE00 0A06 0000 0083"
$'5835 40FE 000A 0600 001B 01B0 1AA0 FE00"
$'0A06 0000 3501 5015 50FE 000A 0600 006A"
$'82A8 2AA8 FE00 0A06 0000 D57D 57D7 F4FE"

```

```

$'000A 0600 01AF AAFA AC1A FE00 0A06 0003"
$'5055 0558 0DFF 000B 0700 06A0 2A02 A80A"
$'80FF 000B 0700 0D60 3603 5415 40FF 000B"
$'0700 0A80 6B06 ABEA C0FF 000B 0700 0D5F"
$'D5FD 5555 40FF 0009 0100 0AFC AA00 C0FF"
$'0009 0100 0DFC 5500 40FF 0009 0100 0FFC"
$'FF00 C0FF 0002 F700 02F7 0002 F700 02F7"
$'0002 F700 02F7 0002 F700 A000 8FA0 0083"
$'FF"
);

```

```

resource 'ICN#' (128) (
( /* array: 2 elements */
/* [1] */
$'0001 0000 0002 8000 0004 4000 0008 2000"
$'0010 1000 0020 0800 0050 0400 0088 0200"
$'0100 0100 0284 0080 0440 0240 0822 0420"
$'1410 0810 220A 1008 4084 3F04 802A 4082"
$'4001 8041 2003 3022 1005 C814 080E 7F8F"
$'0412 3005 0221 0007 0140 8005 0080 6007"
$'0040 1FE5 0020 021F 0010 0407 0008 0800"
$'0004 1000 0002 2000 0001 4000 0000 80"
/* [2] */
$'0001 0000 0003 8000 0007 C000 000F E000"
$'001F F000 003F F800 007F FC00 00FF FE00"
$'01FF FF00 03FF FF80 07FF FFC0 0FFF FFE0"
$'1FFF FFFF 3FFF FFF8 7FFF FFFC FFFF FFFE"
$'7FFF FFFF 3FFF FFFE 1FFF FFFC 0FFF FFFF"
$'07FF FFFF 03FF FFFF 01FF FFFF 0FFF FFFF"
$'007F FFFF 003F FE1F 001F FC07 000F F800"
$'0007 F000 0003 E000 0001 C000 0000 80"
)
);

```

```

resource 'ICN#' (129) (
( /* array: 2 elements */
/* [1] */
$'0FFF FE00 0800 0300 0800 0280 0800 0240"
$'0800 0220 0800 0210 0800 03F8 0801 0008"
$'0880 0008 0801 0208 0840 0008 0801 0408"
$'0820 0008 0801 0808 0810 0008 0801 1008"
$'0AAB AAA8 0809 2008 0804 4008 0803 8008"
$'0800 0008 0801 0008 0800 0008 0801 0008"
$'0800 0008 0801 0008 0800 0008 0801 0008"
$'0800 0008 0800 0008 0800 0008 0FFF FFF8"
/* [2] */
$'0FFF FE00 0FFF FF00 0FFF FF80 0FFF FFC0"
$'0FFF FFE0 0FFF FFF0 0FFF FFF8 0FFE FFF8"
$'0F7F FFF8 0FFE FDF8 0FBF FFF8 0FFF FBF8"
$'0FDD 7FF8 0FFA B7F8 0FE7 DFF8 0FEF FFF8"
$'0D74 5D58 0FB7 DBF8 0F7B BDF8 0EFD 7EF8"
$'0FFF FFF8 0FFF FFF8 0FFF FFF8 0FFF FFF8"
$'0FFF FFF8 0FFF FFF8 0FFF FFF8 0FFF FFF8"
)
);

```

```

(-----)
(----- File UQPlot.MAmake -----)
(-----)

# Copyright © 1989 Charles F. McMath.
# All rights reserved.
#
# This is the makefile. As you can see,
# there's not much to it. Because the application
# files are the 'standard' ones, MABuild will
# figure out (correctly) what needs to be done.

```

```

AppName = QPlot
Creator = 'FGT3'

```



MacApp Workshop

A Tale of Two Quadratic Plotters, Part II



Carl Nelson
Chuck McMath
Hillsborough, VA
Volume 5, Number 9

Another point of view

When I started my efforts, I had to bring the program into the MPW environment, after all I started with someone else's code and wanted to cut and paste as much as I could. The first thing I did was run the source code through the MPW tool 'MapObjects.' This handy tool finds all Pascal UNITS defined in the files you give it to digest, then it goes about finding all PROCEDURES, FUNCTIONS and Objects that are defined. As it encounters the source code it remembers where (file and offset) it saw implementations of the procedures, then it writes out a database (called an ObjectMap) which can then be used by the Browser Desk Accessory to find and view procedures, functions and objects. Essentially you get a list of all Pascal UNITS, in alphabetical order, seen by MapObjects and for every by Pascal UNIT you can see, in alphabetical order, all PROCEDURES and FUNCTIONS contained in the UNIT. You can then click on the name of a PROCEDURE and the DA fetches it for you to view. It is a read-only browser, but you can cut from the displayed text and paste it into your MPW file. (I also pass the latest MPW interfaces through MapObjects. This way I always have them online when I need to cut and paste a Toolbox call.)

Displayed below is the output from doing a 'MapObjects - hvsm' on the original MacTutor Code. Note that where a line has two items on it, the second item is the name of the file containing that unit:

Program

```
QuadraticPlotter plotter
QuadraticPlotter-Private plotter
main
initRects
doUpdates
doMulti
doMouse
doKeyDowns
doActivates
crash
MainEventLoop
InitMyWindow
InitMyMenus
InitMac
```

Unit

```
PlotGlobals PlotGlobals
MyFileStuff myFileStuff
doSaveAs
doSave
```

```
MyPlotStuff myPlotStuff
doQuit
doMenuBar
doGrow
doDrag
doContent
doAbout
Misc Misc
doMessage
MyPrintStuff myPrintStuff
doPrint
doPageSet
MyPrintStuff-Private myPrintStuff
PrintMe
solve solve
solveit
quad
doPlot
PrQDStuff
solve-Private solve
positivecalc
negativecalc
doDialog
PlotMe
```

The Browser turns out to be very helpful in understanding the MacTutor sources. I am not a veteran of previous MacTutor code so I did not know where Dave had decided to hide various parts of the code. I had expected that Dave's code would have some organizing principles that he used to decide where to put PROCEDURES and FUNCTIONS. The article pointed the way but you can also see it from the MapObjects output. The main program is contained in the file 'plotter'. Code to initialize things was found in the Init routines in the main file 'plotter'. What struck me was the pragmatic organization of the code. The principles that organized it seemed straightforward and general enough but I would never have guessed where the functionality would actually be found. I had forgotten how much my familiarity with MacApp and its architecture caused me to make certain assumptions when looking at someone else's code.

For example, when I first received a copy of Chuck's code I wondered if he had taken the time to implement the full page printing option. Knowing MacApp, I knew he would have created a DoSetupMenus and a DoMenuCommand to handle the functionality. So to check his code out I used the MPW search command to find all occurrences of DoSetupMenus in the .inc1.p file he sent and sure enough, I found he had two DoSetupMenus, one in TQPlotApplication and the other in TQPlotView. Selecting the line with TQPlotView yielded exactly what I wanted to

know, it used a field (fOnePage) to place a checkmark beside a couple of menu entries that had associated command names of cPrintWS and cPrintPS which I deduced were probably print window size and print page size which corresponded to the original MacTutor code. Having such strong notions of where things are (and ought to be) makes it easy to maintain and extend someone else's source code. *[Chuck's note: the reason Carl was able to find that section of my code so easily was partially because of the 'MacApp pact' I signed when I converted the code to MacApp—I agreed to place the code (actually I had very few other choices, being a good MacApp citizen and all) which deals with menu selections in the method DoMenuCommand, and Carl knows this.]*

In terms of external design features, the original plot program had two things that caught my attention. First was that the PICT was always drawn to fill the window and the second was that a modal dialog was put up to get the parameters for the equation. I thought that always filling the window was a bit odd but, hey, I'm not really an engineer who plots quadratics all the time (if ever) so I felt that should be left alone. I actually had to do extra work to make the program behave this way. In the MacApp framework the simplest case would have been to setup a fixed size page or view area and then let MacApp scroll the image around for you. The modal dialog was probably an expedient design; there has to be a better user interface. I thought that it would be far better if the plot parameters could be entered in the plot window. My design was simple: put all the fields from the modal dialog into a panel to the side of the plot area, let the user click and type new parameters at any time in a modeless fashion, and then let the user choose PLOT from the menu or press the ENTER key to cause the plot to occur. This simple design change allows multiple windows with their plots and parameters that created them to be showing at the same time.

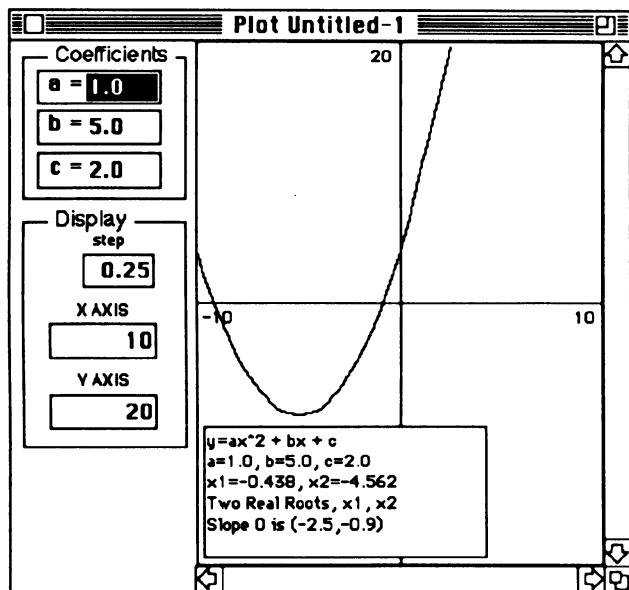


Figure 1.

The original program only allowed the choice of 8 colors for

the plot. I thought as an example program that it should use the new Palette manger and let the user color as many parts of the plot with any color. Also I thought it would be nice to allow for the selection of fonts and typesyles using the hierarchical menus. Besides, I could easily cut and paste all these features from the MacApp samples.

In starting my port of the code I was going to use an external data structure to hold the pertinent data. After I started the design it became obvious that if I did a little bit of encapsulation it would be a good example of the right path to follow and clearly show the advantages of using Objects just like in Chuck's program; I gave the original plot program's data to the appropriate object to manage.

As Chuck pointed out, there are always three methods you will override: TApplication.DoMakeDocument, which creates the appropriate type of Document object; TDocument.DoMakeViews which creates the appropriate type of View and window objects and finally TView.Draw which Draws the contents of a view in your window or on the page.

I had to declare a Document type (TPlotDoc) that DoMakeDocument could bring into existence. This document would hold the plots we would be writing to disk. The files created by this document would be readable with MacDraw and able to be placed into PageMaker. In order to read back these files I implemented a DoRead for TPlotDoc. Granted, it is a simple minded effort but it shows where it should be done. To properly implement writing and reading the plot parameters should also be written and read back.

When a document is created its DoMakeViews method is called to create the views that will display the data in the document. In MacApp 2.0 you can create views by doing a NEW for each view component and setting up the data structures, or you can (preferably) create resources that describe your views and let MacApp read them, create the objects and fill in their data structures. These resources are best created using Apple's ViewEdit. ViewEdit knows about MacApp view resources and lets you build and arrange on screen the view components that will make up your displays. It is indispensable in creating Applications with MacApp 2.0.

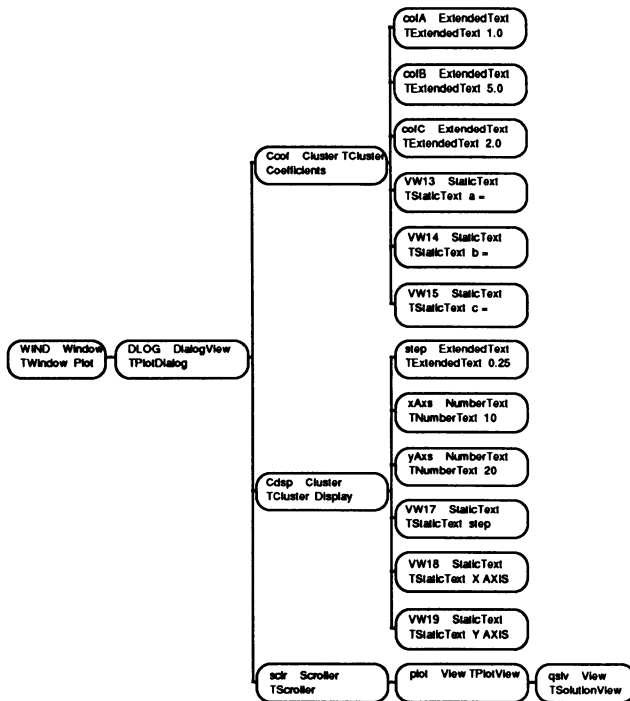
In looking at the view hierarchy, we start with a window which is handled by a standard TWindow. In the window is a dialog handled by TPlotDialog. The dialog contains two display clusters and a scroller. The two clusters contain StaticText and NumberText view items that are used for collecting the equation coefficients and the plot parameters. The scroller contains the plot view which is handled by a TPlotView. The plot view contains a subview for holding the solution text box. It is handled by a TSolutionView.

This hierarchy of views are brought into existence by TPlotDoc's DoMakeViews call to NewTemplateWindow. As each component of the view hierarchy is brought in existence, the associated object which will handle it is found and created by DoCreateViews.

For the TStaticText and TNumberText views MacApp handles all the display and collection of keystrokes. MacApp 2.0B8 does not have facilities for handling NumberViews with

Real numbers, an oversight in the rush to get MacApp 2.0 out the door. Calvin Cock wrote an article (and contributed the code and resources) for publication in the Dec '88 *Frameworks* describing the a new view called TExtendedText which handles real numbers (plug: join the MacApp Developer's Association and receive *Frameworks*). Instead of reinventing the wheel, I used Calvin's ExtendedText Views in the Clusters.

Here is a summary of the view hierarchy which produced the window in Figure 1:



In what I would call a fairly straightforward MacApp approach, TPlotDoc holds the parameters for the plot and TPlotDialog holds the PicHandle generated from the original data structure and does the display of the PICT. TPlotView and TSolutionView do all the work of actually drawing the PICT. Further work could be done on encapsulating and distributing the plotting and display. In particular, PRQDStuff from the original was a fairly large routine that did all the drawing work. I thought it might be proper to split it apart into a controlling object and objects that drew parts of plot. Not to go overboard, but to illustrate the point, I chose to have TPlotDialog create the wrapper and have it call the appropriate component parts that composed the PICT. TPlotDialog sets up the PICT and asks the other objects that it knows draw into it. It should be possible with the framework I have provided to move each AXIS and its labeling into separate views that can be setup independently.

The current design does not utilize any of the MacApp framework to do interactive graphing. If the design of the program changed not to include re-scaling the PICT to the window, it would be possible to create an interaction with the PlotView. For example, you could grab the plot and move it around to show how the coefficients vary.

Given how easy it is to create multiple windows and documents, it would be easy to create plot types and custom windows for different equations. To do this would require that you create new types of plot and view objects then override MacApp's OpenNew method to create the desired new kind of plot document and window.

As Chuck stated MacApp provides a command mechanism for undo and redo. If you look at the menu handling code for changing the plot color and text you will see how I have implemented Undo/Redo by saving and restoring the plot characteristics. I did not go all the way by invalidating the Plot and forcing a redraw instead I, like the Daves, wait until the next plot is drawn. If you look at TPlotDialog.DoKeyCommand you will see how easy it would be to have TPlotDoc.DoMenuCommand force a redraw.

Analysis

Now that we have each discussed our programs, we need to sit back a little and see just what we got out of MacApp. While Carl's version is obviously an improvement given the number of features he added, the benefits of Chuck's version are not so obvious. So Chuck re-takes the floor to convince you he's done a good job.

How much time do we spend doing things?

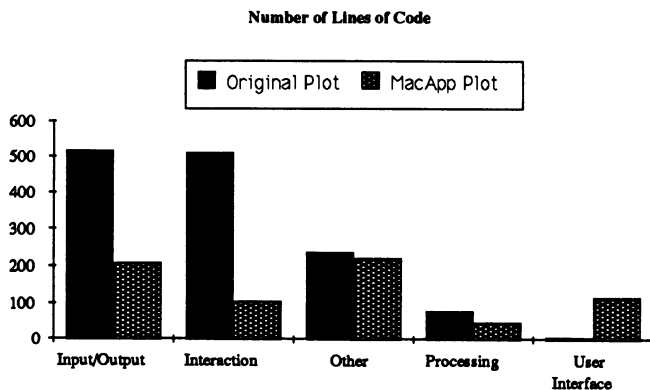
With all of the preceding discussion of my objects, it may seem that we are doing a lot of work in what is admittedly a simple application. Let's see if this is the case. I have analyzed the original and my code to see what percentage of code is used for different functions. This analysis is not perfect in that the original and new versions were written by different people, and therefore it does not allow for different programming styles, but still, the end application is the same (or as similar as a MacApp and non-MacApp application can be), so we can get an idea of where we spent our time. I split the source code into five different categories:

- user interface: the Macintosh look — windows, menu setup and enabling, the 'generic' Mac portion. What code was necessary to make the application look like a Macintosh application? Note that this does not include the code to actually accomplish anything; that's in the next section.
- interaction: the portion of the code that handles any user interaction & event processing. Anything that was written to handle interaction between the user and the application was assigned here.
- input/output routines: any code involved with actually reading or writing data to the disk. This portion also includes code used to print the window, since I considered that an output operation.
- program processing: this portion of code is anything that needed to be written to calculate or determine anything. This code does not have any corresponding on-screen elements.

- initialization and other: this category was used for all pieces of code that I couldn't assign anywhere else.

Chart 1 shows for each category: the number of lines of code written for each version, and Chart 2 the percent of the total code that each section represents (comments and blank lines were removed before the count was made; so this is only counting executable code; 1405 for the original and 705 for MacApp).

Chart 1: Lines of Code comparison: Original vs. MacApp



The first difference that should jump out at you is the number of lines of code written. The MacApp version has half the number of lines as the non-MacApp version. Not only are there fewer lines, but the MacApp version has more features than the original version — it allows multiple windows on the screen and scrolls the window contents, to name two. In addition, the MacApp feature follows Apple's user interface guidelines more closely in the page size/window size feature of the plot. In the original version, selecting 'window size' changes the printed output, but does not change the display in the window — violating the WYSIWYG principle (What You See Is What You Get). So there's another area in which the MacApp program outperforms the original one. Still another area in which the MacApp version improves on the original is in menu handling. The original version has the Edit menu always enabled, even though those menu items do not apply to anything displayed in the window; you can also erroneously try to save or print before you have a plot window on the screen. Again, these operations violate the user interface guidelines — only appropriate menu items should be enabled. Yet the MacApp version does enable all of its menu items correctly. Why? Because of the design of MacApp.

It's interesting to compare how time was spent in creating the two versions (assuming that the code was written at a constant rate. I know this is not an entirely rational assumption, but it's all we have to go on). I've been talking about how MacApp helps you by implementing the standard user interface elements, but looking at the chart you see that the MacApp program has four times as much user interface code (relatively speaking). Does

this mean that I was lying? I sure hope not! The reason for so much user interface code in the MacApp version is that there is a fixed amount of overhead associated with handling menu, mouse, and key events. This overhead is in the form of procedures which must be overridden to provide for the user interface processing. In addition, each object which handles these events must override the appropriate methods to provide the processing. Writing the code for all of these overrides adds up. However, once the procedures are overridden, adding more functionality to the processing is simple and does not require much more code. On the other hand, look at the amount of code we wrote to deal with interaction — the MacApp implement spent only half as much time implementing its interaction code. This is because we build upon the foundation that MacApp provides, and only have to add code to support the extra functionality we wish to provide. MacApp takes care of the normal interaction.

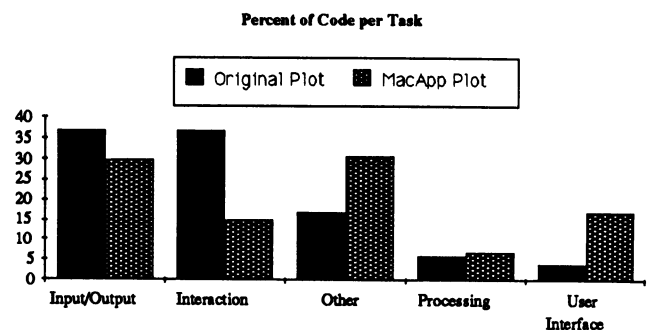


Chart 2: Percent of Code: Original vs. MacApp

This application is somewhat unusual in that the amount of 'pure' processing code is rather small, although it's interesting that the relative amount of processing code is constant. One last thing to note is the amount of code in the 'Other' category. As I categorized the source code for the MacApp version, I realized that much of the code fell into this category. That's because MacApp programs spend a lot of time initializing objects, creating windows and views and documents, and doing a lot of things that are difficult to categorize.

Chart 3 shows the amount of code that came directly from the original version to the MacApp version. The first column is the number of lines of code that was directly cut and pasted from the old version to the new. The second column is the total number of lines in the new category. The third column shows the percentage of each category that the old code makes up. This chart is instructive in that it shows how much of your non MacApp program you will be able to reuse. It shouldn't be surprising that none of the interaction or user interface code was transferable; it's a little more surprising that none of the 'other' code was applicable. If you think about it, however, the MacApp 'other' category is made up of code which is pretty specific to MacApp — object creation and initialization, and the object definitions. This code could not have come from the original program.

	# Lines Taken	Total in New	% of New
Input/Output	181	211	86
Interaction	0	108	0
Other	0	221	0
Processing	48	48	100
User Interface	0	117	0
Totals	229	705	N/A

Chart 3: Code requiring no conversion

As you can see, all of the processing code used in the MacApp program came directly from the original version. This should give some weight to my assertion that in a MacApp program you need to concentrate on the unique portion of your application — for indeed, in the plotter application, the processing is the unique portion. And note also that most of the input/output routines came directly from the original version. Most of this code is related to creating and printing the PICT, and of course remains the same no matter whether you're using MacApp or not.

The bottom line here, though, is that the MacApp version only required 50% as much code as the original version; and fully one-third of that code was lifted directly from the original, which means that the MacApp version only had around 500 new lines of code — one third as much as the original. And this does not yield a program that is crippled or partially functional, it yields a program that conforms to the user interface guidelines, is fully functional (within its design), and is a good bet to run on future Macintosh architectures (not to mention A/UX). This is the big win we get from MacApp: not something for nothing, but more results for less work. Sure, you have to change your thinking, but all I can say is this: look at the bottom line.

Conclusion

So there you have it — two views of a MacApp conversion effort, from two slightly different viewpoints. Although the two MacApp programs may seem quite different, they are much more similar to each other than either is to the original effort. If Carl and I swapped our two MacApp programs and each had to add a feature to them, we would be able to much more easily than if we were to try the same on the original program. All MacApp programs share a structure, an organization; different programs do similar things in the same place. Reusing code from one MacApp program to another is increased. You don't have to spend time searching for the procedure or function you want — you will know where to find it.

We hope you have some idea of the benefits of MacApp and that you'll not only get more done, but you'll get it done more quickly (once you become a MacApp citizen), and your final product will be a real professional job. MacApp is the wave of the future — catch it today! *[Aw Come-on Chuck, knock it off.]*

UPlot.p Plot UNIT FILE

```
([a-,body+,h,o=100,r+,rect+, t=4,u+,#,j=20/57/1$,n+])
( The above is the official MacApp PasMat Style statement. What
you don't use pasmat to make all your source code conform to
your company's (or personal) standard for source code style? )
```

(Copyright © 1986-1988 Apple Computer, Inc.
All rights reserved.)
(Copyright © 1989 by Software Architects, Inc.
All rights reserved.)
(Portions Copyright © 1988 MacTutor
All rights reserved.)

```
{[f-]}
{*
```

This is a very small sample application which uses concepts and program fragments presented in the February 1988 MacTutor Plot Article.

By looking at this program you may be able to gain a better understanding of how to cast a conventional program into the MacApp "Application Framework" or (Class structure depending upon whose jargon you wish to use).

In the tradition of MacApp it defines the basic three Class (object) overrides:

TPlotApplication.DoMakeDocument - Launches appropriate type of Document object.

TPlotDoc.DoMakeViews - Launches appropriate type View and window objects.

TPlotDialog.Draw - Calls its sub components to draw the contents of a view.

TPlotView - To actually plot the thing

TSolveView - To draw the solution text box

In addition it defines commands unique to the plot program. See the text of the accompanying article for details.

```
*)
```

```
{[f+]}
```

```
UNIT UPlot;
```

```
INTERFACE
```

```
USES
```

```
( o MacApp - this includes all of the things
necessary from the MacApp Library )
```

```
UMacApp,
```

```
( o Building Blocks )
```

```
UDialog, UPrinting, UExtendedText,
```

```
( o Implementation Use )
```

```
SANE, ToolUtils, Fonts, Resources,
Script, PickerIntf, Packages;
```

```
CONST
```

```
kSignature = 'Plot';
```

```
( Application signature )
```

```
kFileType = 'PICT';
```

```
( File-type code used for document files
created by this application )
```

```
kPlotDialog = 1010;
```

(MacApp uses a very useful technique for separating menu item numbering from what you want done by choosing a menu item. See MacApp manual pgs xxx-xx for full details. Work is done in MacApp with commands. Each command is assigned a unique number. For example Save is 30 and SaveAs is 32. Numbers below 1200 are reserved for MacApp. In this Application we have chosen a command numbering scheme that makes life easy:

```
Application Commands= 1400
```

```
Font styles = 2000
```

```
Font sizes = 2100
```

```
Font just = 2200
```

```
Font fonts = 2300
```

```
Hierarchical Menus = 2400
```

```
Plot colors = 2500
```

```
)
```

```
( the command to cause a plot to happen )
```

```
cPlotIt = 1401;
```

```
( Command numbers for typestyle attributes )
```

```

cPlainText    = 2001;
cBold         = 2002;
cItalic       = 2003;
cUnderline    = 2004;
cOutline      = 2005;
cShadow       = 2006;
cCondense     = 2007;
cExtend       = 2008;

( Command numbers for font-size commands )
cSizeChange= 2100;
cSizeBase   = 2100;
cSizeMin    = 2100;
cSizeMax    = 2124;

( 2101-2197 reserved for font sizes 1-97 pts. )
cSizeGrow   = 2198;
cSizeShrink = 2199;

( Command numbers cover other stylistic changes )
cJustChange= 2200;
cJustLeft   = 2201;
cJustCenter= 2202;
cJustRight  = 2203;
cFontChange= 2300;

( Command numbers for the hierarchial menu )
cStyle      = 2401;
cSize       = 2402;
cFont       = 2403;
cColor      = 2404;

( Command numbers for changing colors )
cColorChange = 2500;
cColorText   = 2501;
cColorBackground = 2502;
cColorGraph  = 2503;
cColorAxis   = 2504;

( Constant for amount to relative
size text selection )
kRelSizeAmount = 4;

( Constants for the prompts string list )
kPromptsRsrcID = 1001;
kColTextPrompt = 1;
kColBackPrompt = 2;
kColGraphPrompt = 3;
kColAxisPrompt = 4;

( Menu numbers )
mFont = 10;

( Menus displayed on hier. menu system )
kHierDisplayedMBar= 131;

( Menus displayed on non-hier. system )
kNonHierDisplayedMBar = 128;

( Offset added to non-hier menu cmds to get )
kHierMenuOffset = 1000;
( 'view' resource for default values )
kViewRsrcID = 1005;

( PICT comments for our plot and text box )
(selected MacDraw comments)
picDwgBeg    = 130;
picDwgEnd    = 131;
picGrpBeg    = 140;
picGrpEnd    = 141;
TextBegin    = 150;
TextEnd      = 151;
StringBegin  = 151;
StringEnd    = 153;

```

```

TextCenter   = 154;

(postscrip comments)
SetLineWidth = 182;
PostScriptBegin = 190;
TextIsPostscript = 194;
PostScriptEnd = 191;

( The size of a MacDraw Header )
kHeaderSize= 512;

```

TYPE

```

QuadraticType = (NotSolved, RealRoots,
                 SingleSolution, ComplexRoot);

```

(specifications of text display)

```

PlotSpecs = RECORD
theTextFont:   Str255;
theFontNum:    INTEGER;
theTextFace:   Style;
theTextSize:   INTEGER;
theJustification:INTEGER; ( text )
theTextColor:  RGBColor; ( label color )
theGraphColor: RGBColor;
theAxisColor:  RGBColor;
theBackColor:  RGBColor;
END;

```

```

PlotSpecsPtr = ^PlotSpecs;
PlotSpecsHdl = ^PlotSpecsPtr;

```

(Object Definitions)

(-----)

```

TPlotApplication = OBJECT (TApplication)

```

(Initialize application and globals.)

```

PROCEDURE TPlotApplication.IPlotApplication(
itsMainFileType: OSType);

```

(Launches a TPlotDocument)

```

FUNCTION TPlotApplication.DoMakeDocument(
itsCmdNumber: cmdNumber): TDocument;
OVERRIDE;

```

```

PROCEDURE TPlotApplication.IdentifySoftware;
OVERRIDE;

```

```

PROCEDURE TPlotApplication.Fields(
PROCEDURE DoToField(fieldName: Str255;
fieldAddr: Ptr;
fieldType:INTEGER));
OVERRIDE;

```

```

END; ( TPlotApplication )

```

(-----)

```

TPlotDoc = OBJECT (TDocument)

```

```

fPlotDialog: TPlotDialog;
fPlotSpecs : PlotSpecs;
fOldPlot : PicHandle;

```

(coefficients to our quadratic equation)

```

faParam : Real;
fbParam : Real;
fcParam : Real;

```

(plot display parameters)

```

fstepParam : Real;
fxParam : INTEGER;
fyParam : INTEGER;

```

(the solutions to our quadratic)

```

f1stRoot : Real;
f2ndRoot : Real;

```

```

fRootType : QuadraticType;

```



```

    ( setup for the document to hold the plot )
PROCEDURE TPlotDoc.IPlotDocument;

    ( For new doc or revert initial state )
PROCEDURE TPlotDoc.DoInitialState; OVERRIDE;

    ( Generate command to change Look of a plot )
FUNCTION TPlotDoc.DoMakePStyleCmd(
itsStyle:PlotSpecsPtr; itsCmdNumber:CmdNumber): TPStyleCmd;

    ( create all the views needed for document )
PROCEDURE TPlotDoc.DoMakeViews(forPrinting: BOOLEAN);
OVERRIDE;

    ( calculator how much disk space this
    document will need )
PROCEDURE TPlotDoc.DoNeedDiskSpace(
    VAR dataForkBytes, rsrcForkBytes: LONGINT);
OVERRIDE;

    ( read the data for this document )
PROCEDURE TPlotDoc.DoRead(
    aRefNum: INTEGER; rsrcExists,
    forPrinting: BOOLEAN); OVERRIDE;

    ( write the data for this document )
PROCEDURE TPlotDoc.DoWrite(aRefNum: INTEGER;
    makingCopy: BOOLEAN); OVERRIDE;

    ( given a Menu choice handle or pass it on )
FUNCTION TPlotDoc.DoMenuCommand(
    aCmdNumber: cmdNumber): TCommand;
OVERRIDE;

    ( setup the menus for the document )
PROCEDURE TPlotDoc.DoSetupMenus; OVERRIDE;

    ( given the parameters of our
    document calculate a solution )
PROCEDURE TPlotDoc.SolveQuadratic;

PROCEDURE TPlotDoc.ChangeBackColor(
    newColor: RGBColor);

    ($IFC qDebug)
PROCEDURE TPlotDoc.Fields(
    PROCEDURE DoToField(
        fieldName: Str255;
        fieldAddr: Ptr;
        fieldType: INTEGER));

OVERRIDE;
    ($ENDC)

END; ( TPlotDoc )

TPlotView = OBJECT (TView)
fPlotDoc : TPlotDoc;

    ( Add to the pict we will draw on
    the screen or printed page )
PROCEDURE TPlotView.AddToPict(picRect:Rect);

    ( A convenience routine )
PROCEDURE TPlotView.GetQDFrame(
    VAR frameRect:Rect);

    ( The design says the plot fills the
    window, therefore when we are
    resized we must invalidate ourselves)
PROCEDURE TPlotView.Resize(width,
    height: VCoordinate; invalidate: BOOLEAN);
OVERRIDE;
END;

```

```

TSolutionView = OBJECT (TView)
fPlotDoc : TPlotDoc;
fSolveRect : Rect;

( Add to pict we will draw on the screen or printed page )
PROCEDURE TSolutionView.AddToPict(picRect:Rect);
END;

TPlotDialog = OBJECT (TDialogView)
fPlotSize : VPoint;
fPlotView : TPlotView;
fSolutionView: TSolutionView;
fPlotDoc : TPlotDoc;
fPlotPict : PicHandle;

FUNCTION TPlotDialog.DoKeyCommand(
    ch: CHAR; aKeyCode: INTEGER;
    VAR info: EventInfo): TCommand;

OVERRIDE;
FUNCTION TPlotDialog.DoMenuCommand(
    aCmdNumber: CmdNumber):TCommand;
OVERRIDE;

PROCEDURE TPlotDialog.DismissDialog(
    dismitter: IDType; flashDismitter: BOOLEAN);
OVERRIDE;

PROCEDURE TPlotDialog.Draw(area: Rect);
OVERRIDE;

PROCEDURE TPlotDialog.EachSubView(
    PROCEDURE DoToSubView(
        theSubView: TView));
OVERRIDE;

PROCEDURE TPlotDialog.PlotNDrawPict;

PROCEDURE TPlotDialog.GetPlotValues;

PROCEDURE TPlotDialog.DoSetupMenus; OVERRIDE;

END;

TPStyleCmd = OBJECT (TCommand)
fPlotDialog : TPlotDialog;
fOldPlotSpecs: PlotSpecs;
fNewPlotSpecs: PlotSpecs;

    ( Initialize the command; if unsuccessful,
    signalled by Failure mechanism )
PROCEDURE TPStyleCmd.IPStyleCmd(
    itsPlotDialog:TPlotDialog;
    itsNewStyle:PlotSpecsPtr;
    itsCmdNumber: CmdNumber);

PROCEDURE TPStyleCmd.DoIt; OVERRIDE;
PROCEDURE TPStyleCmd.RedoIt; OVERRIDE;
PROCEDURE TPStyleCmd.UndoIt; OVERRIDE;
END;

VAR
gDefaultSpecs: PlotSpecs;
( a Menu Management Global )
gMenuOfs: INTEGER;
( Convenience, fetching string from resource )
gPromptString: Str255;

IMPLEMENTATION
( I M P L E M E N T A T I O N )
(-----)
($S ARes)
( Generally useful routines )

```

```

FUNCTION GetPrompt(index: INTEGER): StringPtr;
BEGIN
  GetIndString(gPromptString,
               kPromptsRsrcID, index);
  GetPrompt := @gPromptString;
END;

FUNCTION Real2Str(aReal: Real; theDigits: INTEGER): Str255;
VAR aStr: DecStr;
    form: DecForm;
BEGIN
  form.style := FixedDecimal;
  form.digits := theDigits;
  Num2Str(form, aReal, aStr);

  Real2Str := aStr;
END;

($IFC qDebug)
($IFC qTrace) ($D+) ($ENDC)
( In the final version of MacApp 2.0 there will some kind of
  support for REAL numbers in text entry fields, for now we use
  Calvins Cock's code from the Dec '88 Frameworks )

PROCEDURE MyFieldToString(theData: Ptr;
                           fieldType: integer; VAR theString: str255);
CONST
  DecPrec = 2;
  ( Change if you want more decimal precision )
TYPE
  TAlias = RECORD
    CASE integer OF
      bReal, bSingle: (asReal: Real);
      bDouble: (asDouble: Double);
      bExtended: (asExtended: Extended);
    END;
VAR
  alias: TAlias;
  aDecForm: DecForm;
  x: Extended;
  NumStr: DecStr;
BEGIN
  alias := Pointer(theData);
  WITH alias DO
    CASE fieldType OF
      bReal, bSingle:
        BEGIN
          aDecForm.style := FixedDecimal;
          aDecForm.digits := DecPrec;
          x := asReal;
          Num2Str(aDecForm, x, NumStr);
          theString := str255(NumStr);
        END;
      bDouble:
        BEGIN
          aDecForm.style := FixedDecimal;
          aDecForm.digits := DecPrec;
          x := asDouble;
          Num2Str(aDecForm, x, NumStr);
          theString := str255(NumStr);
        END;
      bExtended:
        BEGIN
          aDecForm.style := FixedDecimal;
          aDecForm.digits := DecPrec;
          x := asExtended;
          Num2Str(aDecForm, x, NumStr);
          theString := str255(NumStr);
        END;
      OTHERWISE StdFieldToString(theData,
                                fieldType, theString);
    END;
END;

```

```

($IFC qTrace) ($D++) ($ENDC)
($ENDC qDebug)

(-----)
($S AheadFile)
PROCEDURE ReadBytes(theRefNum: INTEGER;
                    size: LONGINT; buffer: Ptr);
( Utility for reading data from a file )

BEGIN
  FailOSErr(FSRead(theRefNum, size, buffer));
END;

(-----)
($S AWriteFile)
PROCEDURE WriteBytes(theRefNum: INTEGER;
                     size: LONGINT; buffer: Ptr);
( Utility for writing data to a file. )
BEGIN
  FailOSErr(FSWrite(theRefNum, size, buffer));
END;

(-----)
($S AInit)

PROCEDURE TPlotApplication.IPlotApplication(
                                itsMainFileType: OSType);

VAR
  fontName: Str255;
  aTEView: TTEView;

BEGIN

  ( qNeedsHierarchicalMenus is a MacApp compile time flag you
    can set which will require the use of Heirarchical Menus )

  ($IFC NOT qNeedsHierarchicalMenus)
  IF NOT gConfiguration.hasHierarchicalMenus THEN
    BEGIN
      gMBarDisplayed := kNonHierDisplayedMBar;
      gMenuOfs := 0;
    END
  ELSE
    ($ENDC)
    BEGIN
      gMBarDisplayed := kHierDisplayedMBar;
      gMenuOfs := kHierMenuOffset;
    END;

  IApplication(itsMainFileType);

  ( Do not setup the menus if we were started up with the
    request to print )
  IF NOT gFinderPrinting THEN
    BEGIN
      AddResMenu(GetMHandle(mFont), 'FONT');

      SetStyle(cBold, [bold]);
      SetStyle(cUnderline, [underline]);
      SetStyle(cItalic, [italic]);
      SetStyle(cOutline, [outline]);
      SetStyle(cShadow, [shadow]);
      SetStyle(cCondense, [condense]);
      SetStyle(cExtend, [extend]);
    END;

  ( This is an example of questional code reuse. We know that
    TTEViews holds lots of info about fonts, sizes and color and
    we have the Viewedit tool at our disposal, so why not use it
    to create a useful resource to get our initial values from
    instead of hard-wiring the defaults. The technique here is to
    define a TTEView resource and use all those great fields that
    you can setup with viewEdit as the defaults, then steal the

```

values from the TTEVIEW object and trash the TTEview once all the work is done.)

```

(fetch the resource)
aTEView := TTEView(DoCreateViews(NIL, NIL,
                                kViewRsrcID, gZeroVPt));
FailNIL(aTEView);

( Set up initial text specs )
GetFontName(aTEView.fTextStyle.tsFont,
            fontName);

WITH gDefaultSpecs, aTEView DO
BEGIN
    theTextFont := fontName;
    theTextFace := fTextStyle.tsFace;
    theTextSize := fTextStyle.tsSize;
    theTextColor := fTextStyle.tsColor;
    theJustification := fJustification;
    theBackColor := gRGBWhite;
END;
aTEView.Free;

( Until MacApp 2.0 debug and ViewTemplates support REALs
this is our work around from Calvin Cock's January '89 MapApp
Frameworks article. )
gFieldToStrRtn := @MyFieldToString;

END;

(-----)
($IFC qDebug)
($S ADebug)
PROCEDURE TPlotApplication.IdentifySoftware;

BEGIN
    WriteLn('Plot Source date: 31 Jan 89;Compiled:',
            COMPDATE, ' @ ', COMPTIME);
    INHERITED IdentifySoftware;
END;

PROCEDURE TPlotApplication.Fields(
    PROCEDURE DoToField(fieldName: Str255;
                        fieldAddr: Ptr;
                        fieldType: INTEGER));

    OVERRIDE;
BEGIN
    WITH gDefaultSpecs DO
    BEGIN
        DoToField(' Font', @theTextFont, bFontName);
        DoToField(' Face', @theTextFace, bStyle);
        DoToField(' Size', @theTextSize, bInteger);
    END;
END;
($ENDC)

(*****
( T P l o t D o c u m e n t )
(*****
(-----)
($S AOpen)

PROCEDURE TPlotDoc.IPlotDocument;
VAR
    aRect:      Rect;
BEGIN
    IDocument(kFileType, kSignature,
               kUsesDataFork, NOT kUsesRsrcFork,
               NOT kDataOpen, NOT kRsrcOpen);
    fOldPlot := NIL;
END;

(-----)
($S AOpen)
FUNCTION TPlotApplication.DoMakeDocument(itsCmdNumber:

```

```

cmdNumber): TDocument;
VAR
    aPlotDocument:  TPlotDoc;
    dimensions:      Rect;
BEGIN
    ( Allocate and initialize the document )
    NEW(aPlotDocument);
    FailNIL(aPlotDocument);
    aPlotDocument.IPlotDocument;
    DoMakeDocument := aPlotDocument;
END;

(-----)
($S AOpen)
PROCEDURE TPlotDoc.DoInitialState; OVERRIDE;
BEGIN
    ( when reverting to an old copy of a document we
      need to reset to some reasonable values )
    fPlotSpecs := gDefaultSpecs;
    fOldPlot := NIL;
END;

(-----)
FUNCTION TPlotDoc.DoMakePStyleCmd(itsStyle:PlotSpecsPtr;
                                itsCmdNumber:CmdNumber): TPStyleCmd ;
VAR
    aPStyleCmd:TPStyleCmd;
    aPlotDialog: TPlotDialog;
BEGIN
    New(aPStyleCmd);
    FailNIL(aPStyleCmd);
    aPlotDialog := fPlotDialog;
    aPStyleCmd.IPStyleCmd(aPlotDialog, itsStyle, itsCmdNumber);
    DoMakePStyleCmd := aPStyleCmd;
END;

(-----)
($S AOpen)
PROCEDURE TPlotDoc.DoMakeViews(forPrinting: BOOLEAN);
VAR
    aWindow:      TWindow;
    aPlotDialog:  TPlotDialog;
    aPlotView:    TPlotView;
    aTEView:      TTEView;
    aSolutionView: TSolutionView;
    aCluster:      TCluster;
    aPrintHandler: TStdPrintHandler;
    anExtendedText: TExtendedText;
    aRect:         Rect;
BEGIN
    ( We want to dynamically call these in
      exisitance. Do a New() call for each object
      type, so the linker doesn't strip them out. )
    IF gCreateWithTemplates THEN
    BEGIN
        New(aPlotDialog);
        New(aPlotView);
        New(aTEView);
        New(aSolutionView);
        New(aCluster);
        New(anExtendedText);
    END;

    ( we will now ceate and connect up all the view
      and document variables )

    aWindow := NewTemplateWindow(kPlotDialog, SELF);
    ( bring our plot window into exisitance )

    aPlotDialog := TPlotDialog(aWindow.FindSubView('DLOG'));
    FailNIL(aPlotDialog);
    (find dialog portions & make certian we have it )

```

```

fPlotDialog := aPlotDialog;
( save this away for late when we need
  to easily find the dialog )

aPlotDialog.fPlotDoc := SELF;
( and of course we need to cross refer so
  tell the plotdialog who its doc is )

IF (fOldPlot <> NIL)
| (GetHandleSize(Handle(fOldPlot)) > 0) THEN
  aPlotDialog.fPlotPICT := fOldPlot
  ( If the document has an old plot PICT display it )
ELSE
  aPlotDialog.GetPlotValues;
  ( Use the values from the resource to
    generate the first plot )

  ( Find the PlotView and make some
    connections to our PlotDocument )
  aPlotView := TPlotView(aWindow.FindSubView('plot'));
  FailNIL(aPlotView);
  aPlotView.fPlotDoc := SELF;
  aPlotDialog.fPlotView := aPlotView;
  aPlotDialog.fPlotPICT := NIL;
  aPlotDialog.fPlotSize := aPlotView.fSize;

  ( Find the SolutionView and make some
    connections to our PlotDocument )
  aSolutionView := TSolutionView(aWindow.FindSubView('qslv'));
  FailNIL(aSolutionView);
  aSolutionView.fPlotDoc := SELF;
  aPlotDialog.fSolutionView := aSolutionView;

  ( while we are at it, retrieve the rectangle
    size we will use to display in )
  SetRect(aRect, 0, 0, aSolutionView.fSize.h,
    aSolutionView.fSize.v);
  aSolutionView.fSolveRect := aRect;

  ( we want to limit the minimum size this
    window can be so... use the size of our
    clusters to determine the minimum )
  aCluster := TCluster(aWindow.FindSubView('Ccof'));
  aWindow.fResizeLimits.top := aCluster.fSize.v*2;
  aWindow.fResizeLimits.left := aCluster.fSize.h*3;
  aCluster := TCluster(aWindow.FindSubView('Cdsp'));
  aWindow.fResizeLimits.top := aWindow.fResizeLimits.top
    + aCluster.fSize.v;

NEW(aPrintHandler);
FailNIL(aPrintHandler);
aPrintHandler.IStdPrintHandler(SELF, ( its document )
  aPlotDialog, ( its view )
  ( does not have square dots )
  FALSE, ( horizontal page size is fixed )
  TRUE, ( vertical page size is variable
  (could be set to true on
  non-style TE systems) )
  FALSE);
aPrintHandler.fMinimalMargins := FALSE;
END;

(-----)
($S ASetCommand)
FUNCTION TPlotDoc.DoMenuCommand(
  aCmdNumber: CmdNumber): TCommand; OVERRIDE;
VAR
  aName: Str255;
  menu: INTEGER;
  item: INTEGER;
  newStyle: PlotSpecs;

(-----)
PROCEDURE DoSizeChange(base: CmdNumber);

```

```

BEGIN
  newStyle.theTextSize := aCmdNumber - base;
  DoMenuCommand := DoMakePStyleCmd(
    @newStyle, cSizeChange);
END;

(-----)
PROCEDURE DoRelSizeChange(amount: INTEGER);
BEGIN
  WITH newStyle DO
    theTextSize := theTextSize + amount;
    DoMenuCommand := DoMakePStyleCmd(@newStyle, cSizeChange);
END;

(-----)
PROCEDURE DoFontChange;
BEGIN
  GetItem(GetMHandle(menu), item, newStyle.theTextFont);
  GetFNum(aName, newStyle.theFontNum);
  DoMenuCommand := DoMakePStyleCmd(@newStyle, cFontChange);
END;

(-----)
PROCEDURE DoColTextChange;
VAR
  aColor: RGBColor;
BEGIN
  aColor := fPlotSpecs.theTextColor;
  IF GetColor(Point($00400040),
    GetPrompt(kColTextPrompt)~,
    aColor, newStyle.theTextColor) THEN
    DoMenuCommand := DoMakePStyleCmd(
      @newStyle, cColorText);
END;

(-----)
PROCEDURE DoColGraphChange;
VAR
  aColor: RGBColor;
BEGIN
  aColor := fPlotSpecs.theGraphColor;
  IF GetColor(Point($00400040),
    GetPrompt(kColGraphPrompt)~, aColor,
    newStyle.theGraphColor) THEN
    DoMenuCommand := DoMakePStyleCmd(@newStyle, cCol-
orGraph);
END;

(-----)
PROCEDURE DoColAxisChange;
VAR
  aColor: RGBColor;
BEGIN
  aColor := fPlotSpecs.theAxisColor;
  IF GetColor(Point($00400040),
    GetPrompt(kColAxisPrompt)~, aColor,
    newStyle.theAxisColor) THEN
    DoMenuCommand := DoMakePStyleCmd(
      @newStyle, cColorAxis);
END;

(-----)
PROCEDURE DoColBackChange;
VAR
  aColor: RGBColor;
BEGIN
  aColor := fPlotSpecs.theBackColor;
  IF GetColor(Point($00400040),
    GetPrompt(kColBackPrompt)~, aColor,
    newStyle.theBackColor) THEN
    BEGIN
      DoMenuCommand := DoMakePStyleCmd(
        @newStyle, cColorBackGround);
    END;
END;

```

```

END;

(-----)
PROCEDURE DoJustChange;
VAR
    newJust: INTEGER;
BEGIN
    CASE aCmdNumber OF
        cJustLeft:
            newJust := teJustLeft;
        cJustCenter:
            newJust := teJustCenter;
        cJustRight:
            newJust := teJustRight;
    END;
    newStyle.theJustification := newJust;
    DoMenuCommand := DoMakePStyleCmd(@newStyle, aCmdNumber);
END;

(-----)
PROCEDURE DoPlainChange;
BEGIN
    newStyle.theTextFace := [];
    DoMenuCommand := DoMakePStyleCmd(@newStyle, cStyleChange);
END;

(-----)
PROCEDURE DoStyleChange;
VAR
    newFace: Style;
BEGIN
    WITH newStyle DO
        BEGIN
            CASE aCmdNumber OF
                cBold:
                    newFace := [bold];
                cItalic:
                    newFace := [italic];
                cUnderline:
                    newFace := [underline];
                cOutline:
                    newFace := [outline];
                cShadow:
                    newFace := [shadow];
                cCondense:
                    newFace := [condense];
                cExtend:
                    newFace := [extend];
            END;
            IF newFace * theTextFace = newFace THEN
                theTextFace := theTextFace - newFace
            ELSE
                theTextFace := theTextFace + newFace;
            END;
            DoMenuCommand := DoMakePStyleCmd(
                @newStyle, cStyleChange);
        END;
    END;

(-----)
BEGIN
    DoMenuCommand := gNoChanges;

    newStyle := fPlotSpecs;

    CmdToMenuItem(aCmdNumber, menu, item);

    IF menu = mFont THEN
        DoFontChange
    ELSE
        CASE aCmdNumber OF
            cSizeMin..cSizeMax:
                DoSizeChange(cSizeBase);
            cSizeGrow:
                DoRelSizeChange(kRelSizeAmount);
        END;
    END;

```

```

cSizeShrink:
    DoRelSizeChange(- kRelSizeAmount);
cJustLeft..cJustRight:
    DoJustChange;
cPlainText:
    DoPlainChange;
cBold..cExtend:
    DoStyleChange;
cColorText:
    DoColTextChange;
cColorGraph:
    DoColGraphChange;
cColorAxis:
    DoColAxisChange;
cColorBackground:
    DoColBackChange;

    OTHERWISE
        DoMenuCommand := INHERITED
            DoMenuCommand(aCmdNumber);

END;
END;

```

```

(-----)
($S ARes)
PROCEDURE TPlotDoc.DoSetupMenus; OVERRIDE;
VAR
    hasColor: BOOLEAN;
    hasStyle: BOOLEAN;
    checkPlain: BOOLEAN;
    checkSize: BOOLEAN;
    checkFont: BOOLEAN;
    specChange: BOOLEAN;
    just: INTEGER;
    item: INTEGER;
    fnt: INTEGER;
    c: INTEGER;
    aMode: INTEGER;
    aFace: Style;
    aMenuHandle: MenuHandle;
    aName: Str255;
    aStyle: TextStyle;
    theFont: INTEGER;
    aStr255: Str255;
BEGIN
    INHERITED DoSetupMenus;

    hasColor := gConfiguration.hasColorQD;
    hasStyle := gConfiguration.hasStyleTextEdit;

    aStr255 := fPlotSpecs.theTextFont;
    GetFNum(aStr255, aStyle.tsFont);
    WITH aStyle, fPlotSpecs DO
        BEGIN
            tsFace := theTextFace;
            tsSize := theTextSize;
            tsColor := theTextColor;
        END;
        checkPlain := aStyle.tsFace = [];
        checkFont := TRUE;

        aMenuHandle := GetMHandle(mFont);

        GetFontName(aStyle.tsFont, aName);
        ( Get real font number in case tsFont is )
        GetFNum(aName, theFont);
        ( ...the system or application font. )
        FOR item := 1 TO CountMItems(aMenuHandle) DO
            BEGIN
                ( There can be more than 31 menu entries
                  with scrolling menus, but trying to enable
                  an item with number > 31 is bad news.
                  If the menu itself is enabled (which it
                  will be in MacApp if any of the first 31

```

```

items is enabled), then the extras
will always be enabled.)
IF item <= 31 THEN
    EnableItem(aMenuHandle, item);
IF checkFont THEN
    BEGIN
        GetItem(aMenuHandle, item, aName);
        GetFNum(aName, fnt);
        CheckItem(aMenuHandle, item, fnt = theFont);
    END;
END;

just := fPlotSpecs.theJustification;
( Enable justification related menu items )
EnableCheck(cJustLeft, TRUE, (just = teJustLeft));
EnableCheck(cJustCenter, TRUE, (just = teJustCenter));
EnableCheck(cJustRight, TRUE, (just = teJustRight));

($IFC NOT qNeedsHierarchicalMenus)
IF gConfiguration.hasHierarchicalMenus THEN
($ENDC)
    BEGIN
        Enable(cStyle, TRUE);      ( Enable sub-menus )
        Enable(cSize, TRUE);
        Enable(cFont, TRUE);
        Enable(cColor, hasColor);
    END;

aFace := aStyle.tsFace;
EnableCheck(cPlainText, TRUE, checkPlain);
( Enable normal Style menu items )
EnableCheck(cBold, TRUE, bold IN aFace);
EnableCheck(cItalic, TRUE, italic IN aFace);
EnableCheck(cUnderline, TRUE, underline IN aFace);
EnableCheck(cOutline, TRUE, outline IN aFace);
EnableCheck(cShadow, TRUE, shadow IN aFace);
EnableCheck(cCondense, TRUE, condense IN aFace);
EnableCheck(cExtend, TRUE, extend IN aFace);

FOR c := cSizeMin TO cSizeMax DO
    BEGIN
        IF hasStyle THEN
            checkSize := FALSE
        ELSE
            checkSize := (c - cSizeBase) = aStyle.tsSize;
            EnableCheck(c, TRUE, checkSize);

        IF ((NOT hasStyle) |
            ( If the record isn't styled, or )
            RealFont(aStyle.tsFont, c-cSizeBase))
            ( the size is a real one )
            THEN aFace := [outline]
            ( ...then we outline it )
        ELSE
            aFace := [];
            SetStyle(c, aFace);
        END;

        Enable(cSizeGrow, TRUE);
        Enable(cSizeShrink, TRUE);

        Enable(cColorText, hasColor);
        Enable(cColorBackground, hasColor);
        Enable(cColorGraph, hasColor);
        Enable(cColorAxis, hasColor);

    END;

(-----)
($S AWriteFile)
PROCEDURE TPlotDoc.DoNeedDiskSpace(
    VAR dataForkBytes, rsrcForkBytes: LONGINT);
VAR
    r: Rect;

```

```

BEGIN
( In other MacApp Samples we would normally get
the Print record space requirements by doing:

    INHERITED DoNeedDiskSpace(dataForkBytes, rsrcForkBytes);

    BUT seeing as we are trying to imitate a
    MacDraw File we will just do our size
    calculation
)

dataForkBytes := kHeaderSize ( for std MacDraw Header )
    + GetHandleSize(Handle(fPlotDialog.fPlotPICT));

( For now we will write only the PICT Handle in
the future we might want to write the
PlotSpecs and other parameters to the resource
fork of the file )
rsrcForkBytes := 0;
END;

(-----)
($S AReadFile)

PROCEDURE TPlotDoc.DoRead(aRefNum: INTEGER; rsrcExists,
forPrinting: BOOLEAN);
VAR
    fi: FailInfo;
    aPICTSize: LONGINT;
    aPICTHandle: Handle;

PROCEDURE SkipDocHeaderInfo;
    BEGIN
        ( skip the dummy header )
        FailOSErr( SetFPos(aRefNum, fsFromStart, kHeaderSize));
    END;

PROCEDURE Hd1ReadFailure(error: OSErr; message: LONGINT);
    BEGIN
        ( We ran into trouble reading the data for
        now do nothing )
    END;

BEGIN
    CatchFailures(fi, Hd1ReadFailure);

    ( Normally, we would ask our ancestors to read
    in their data by:
    INHERITED DoRead(aRefNum, rsrcExists, forPrinting);
    but we are imitating a MacDraw Doc and have
    to behave like one
    )

    SkipDocHeaderInfo;

    IF fOldPlot <> NIL THEN
        DisposHandle(Handle(fOldPlot));

        FailOSErr(GetEOF(aRefNum, aPICTSize));
        aPICTSize := aPICTSize - kHeaderSize;
        aPICTHandle := NewHandle(aPICTSize);

        HLock(aPICTHandle);
        ReadBytes(aRefNum, aPICTSize, aPICTHandle^);
        HUnlock(aPICTHandle);

        fOldPlot := PicHandle(aPICTHandle);

        Success(fi);
    END;

(-----)
($S AWriteFile)
PROCEDURE TPlotDoc.DoWrite(aRefNum: INTEGER; makingCopy:

```

```

BOOLEAN);

PROCEDURE WriteHeaderInfo;
VAR
  aPtr: Ptr;
BEGIN
  aPtr := NewPtrClear(kHeaderSize);
  FillNil(aPtr);
  WriteBytes(aRefNum, kHeaderSize, aPtr);
  DisposPtr(aPtr);
END;

PROCEDURE WritePlotInfo;
VAR
  aPICHandle: Handle;
  aPICSize: LONGINT;
BEGIN
  aPICHandle := Handle(fPlotDialog.fPlotPict);
  aPICSize := GetHandleSize(aPICHandle);

  HLock(aPICHandle);
  WriteBytes(aRefNum, aPICSize, aPICHandle);
  HUnlock(aPICHandle);
END;

BEGIN
  WriteHeaderInfo;
  WritePlotInfo;
END;

(-----)
($S ARes)
PROCEDURE TPlotDoc.SolveQuadratic;
VAR
  a,b,c,check: real;

  FUNCTION PositiveCalc(a, b, check:real):real;
  BEGIN
    PositiveCalc := (-b + sqrt(check)) / (2 * a);
  END;

  FUNCTION NegativeCalc(a, b, check:real):real;
  BEGIN
    NegativeCalc := (-b - sqrt(check)) / (2 * a);
  END;

BEGIN
  a := faParam;
  b := fbParam;
  c := fcParam;

  check := (b * b) - (4 * a * c);

  IF check = 0 THEN
    ( we have a double root (same place twice) )
    BEGIN
      fRootType := SingleSolution;
      f1stRoot := PositiveCalc(a, b, check);
      f2ndRoot := f1stRoot;
    END

  ELSE IF check > 0 THEN
    ( we have a pair of "real" x axis crossings )
    BEGIN
      fRootType := RealRoots;
      f1stRoot := PositiveCalc(a, b, check);
      f2ndRoot := NegativeCalc(a, b, check);
    END

  ELSE IF check < 0 THEN
    ( roots are represented by complex number )
    BEGIN
      fRootType := ComplexRoot;
      check := -check;

      f1stRoot := PositiveCalc(a, b, check);
      f2ndRoot := NegativeCalc(a, b, check);
    END;

    (-----)
    ($S ANonRes)
    PROCEDURE TPlotDoc.ChangeBackColor(newColor: RGBColor);
    VAR
      oldPort: GrafPtr;
      itsWindow: TWindow;
    BEGIN
      itsWindow := TView(fPlotDialog).GetWindow;
      IF itsWindow <> NIL THEN
        BEGIN
          GetPort(oldPort);
          SetPort(itsWindow.fWMgrWindow);
          RGBBackColor(newColor);
          itsWindow.ForceRedraw;
          SetPort(oldPort);
        END;
      END;

    (-----)
    ($IFC qDebug)
    ($S AFields)
    PROCEDURE TPlotDoc.Fields(
      PROCEDURE DoToField(fieldName:
        Str255; fieldAddr: Ptr; fieldType: INTEGER); OVERRIDE;
    VAR
      aStr: DecStr;
      form: DecForm;
      aReal: Real;
    BEGIN
      form.style := FloatDecimal;
      form.digits := 6;

      DoToField('TPlotDoc', NIL, bClass);
      aReal := faParam;
      Num2Str(form, aReal, aStr);
      DoToField('faParam', @aStr, bString);

      aReal := fbParam;
      Num2Str(form, aReal, aStr);
      DoToField('fbParam', @aStr, bString);

      aReal := fcParam;
      Num2Str(form, aReal, aStr);
      DoToField('fcParam', @aStr, bString);

      aReal := fstepParam;
      Num2Str(form, aReal, aStr);
      DoToField('fstepParam', @aStr, bString);
      DoToField('fxParam', @fxParam, bInteger);
      DoToField('fyParam', @fyParam, bInteger);

      DoToField('Font', @fPlotSpecs.theTextFont, bFontName);
      DoToField('Face', @fPlotSpecs.theTextFace, bStyle);
      DoToField('Size', @fPlotSpecs.theTextSize, bInteger);

      INHERITED Fields(DoToField);
    END;
  ($ENDC)

  (*****
  ( T P l o t D i a l o g )
  (*****
  ($S )
  PROCEDURE TPlotDialog.DismissDialog(
    dismisser: IDType; flashDismisser: BOOLEAN);
  OVERRIDE;
  BEGIN
    INHERITED DismissDialog(dismisser, flashDismisser);
  END;

```

```

(-----)
FUNCTION TPlotDialog.DoKeyCommand(ch: CHAR;
aKeyCode: INTEGER; VAR info: EventInfo): TCommand;
OVERRIDE;
BEGIN
( If Enter is pressed, assume we have new
parameters and a new plot to create. )
IF (ch = chEnter) THEN
BEGIN
(we have everything we need, lets DOIT!)
PlotNDrawPICT;
(We did all the work no reason to
generate a command)
DoKeyCommand := gNoChanges; END
ELSE
DoKeyCommand := INHERITED
DoKeyCommand(ch, aKeyCode, info);
END;

(-----)
FUNCTION TPlotDialog.DoMenuCommand(
aCmdNumber: CmdNumber): TCommand; OVERRIDE;
BEGIN
( Plot Menu Command was chosen so create plot. )
IF (aCmdNumber = cPlotIt) THEN
BEGIN
PlotNDrawPICT; ( we have everything, lets DOIT! )
DoMenuCommand := gNoChanges;
( We did all the work no reason to generate a command )
END
ELSE
DoMenuCommand := INHERITED DoMenuCommand(aCmdNumber);
END;

(-----)
PROCEDURE TPlotDialog.DoSetupMenus; OVERRIDE;
BEGIN
Enable(cPlotIt, TRUE);
INHERITED DoSetupMenus;
END;

(-----)
PROCEDURE TPlotDialog.Draw(area: Rect); OVERRIDE;
VAR
plotRect : Rect;
dontCare : BOOLEAN;
BEGIN
dontCare := fPlotView.Focus;
( set the world up to focus all drawing
in the PlotView )

fPlotView.GetQDFrame(plotRect);
( the design says we draw into the
available frame )

DrawPicture(fPlotPICT, plotRect);
( Draw the PICT in the PlotView )

dontCare := SELF.Focus;
( shift the focus back to us )
END;

(-----)
PROCEDURE TPlotDialog.EachSubView(PROCEDURE
DoToSubView(theSubView: TView));
BEGIN
( We do not want all of our clever
dialog elements printing so lets
skip them when printing )
IF NOT gPrinting THEN
INHERITED EachSubView(DoToSubView);
END;

(-----)

```

```

PROCEDURE TPlotDialog.PlotNDrawPICT;
VAR
plotRect : Rect;
solveRect : Rect;
saveClip : RgnHandle;
pstate : PenState;
dontCare : BOOLEAN;
aFontNum : INTEGER;
BEGIN
GetPlotValues;
( get current values from the view )
fPlotDoc.SolveQuadratic;
( for the parameters retrieved, setup
the solution values )

IF fPlotPICT <> NIL THEN
KillPicture(fPlotPICT);

dontCare := fPlotView.Focus;
fPlotView.GetQDExtent(plotRect);
( change MacApp's focus to the plotview
to properly get our plot rect )
dontCare := SELF.Focus;
( and of course lets set FOCUS back to us.
The above process of setting up plotRect
might have been better handled by
declaring an fRect in plotview
and having the resize method of plotview
keep it up to date )

fPlotPICT := OpenPicture(plotRect);

saveClip := NewRgn;
FailNIL(SaveClip);
GetClip(SaveClip);

GetPenState(pstate);

( setup the font info for our plot )
($Push) ($H-)
GetFNum(fPlotDoc.fPlotSpecs.theTextFont, aFontNum);
($H+)
TextFont(aFontNum);
TextSize(fPlotDoc.fPlotSpecs.theTextSize);
TextFace(fPlotDoc.fPlotSpecs.theTextFace);
( we also have: theTextColor,
theJustification, theBackColor
that we might want to do something with
in the future )
TextMode(srcOr);

(Begin MacDraw Document PICT)
PicComment(picDwgBeg, 0, NIL);
PicComment(picGrpBeg, 0, NIL);

fPlotView.AddToPict(plotRect);

solveRect := fSolutionView.fSolveRect;
( get our plot rect, we know that this
is a fixed rect )

offsetRect(solveRect, 4(border),
plotRect.Bottom-(solveRect.Bottom+4(border)));
( slide the textbox on down to the
bottom of the view area this assumes
that the solution textbox is smaller
than the plotRect )

fSolutionView.AddToPict(solveRect);

PicComment(picGrpEnd, 0, NIL);
PicComment(picDwgEnd, 0, NIL);

```



```

ClosePicture;

SetPenState(pstate);
SetClip(saveClip);

DisposeRgn(saveClip);

ForceRedraw;
( make the dialog rect invalid )
END;

(-----)
PROCEDURE TPlotDialog.GetPlotValues;
BEGIN
    fPlotDoc.faParam :=
        TExtendedText(FindSubView('cofA')).GetValue;
    fPlotDoc.fbParam :=
        TExtendedText(FindSubView('cofB')).GetValue;
    fPlotDoc.fcParam :=
        TExtendedText(FindSubView('cofC')).GetValue;

    fPlotDoc.fstepParam :=
        TExtendedText(FindSubView('step')).GetValue;
    fPlotDoc.fxParam :=
        TNumberText(FindSubView('xAxis')).GetValue;
    fPlotDoc.fyParam :=
        TNumberText(FindSubView('yAxis')).GetValue;
END;

(-----)
PROCEDURE TPlotView.GetQDFrame(VAR frameRect:Rect);
VAR
    extent:    VRect;
BEGIN
    GetFrame(extent);
    ViewToQDRect(extent, frameRect);
END;

(-----)
PROCEDURE TPlotView.AddToPict(picRect:Rect);
CONST
    AxisLabelIndent = 4;
TYPE
    Widpt = Point;
    WidPtr = ^WidPt;
    WidHdl = ^WidPtr;
VAR
    x, y      : Real;
    a,b,c     : Real;
    step      : Real;
    halfXScale : Real;
    xPictScale : Real;
    yPictScale : Real;

    lastPt    : Point;
    thisPt    : Point;
    xScale    : INTEGER;
    yScale    : INTEGER;
    CenterHorz : INTEGER;
    CenterVert : INTEGER;
    PictHorz   : INTEGER;
    PictVert   : INTEGER;

    leading    : INTEGER;
    fInfo      : FontInfo;
    Width      : WidHdl;
    axisLabel  : str255;

    newColor   : RGBColor;
    oldBack    : RGBColor;
    oldFore    : RGBColor;

    drawLine   : BOOLEAN;

```

```

BEGIN
    Width := WidHdl(NewHandle(sizeof(widpt)));
    Width^.h := 10;
    Width^.v := 1;

    GetFontInfo(fInfo);
    leading := fInfo.descent
        + fInfo.ascent + fInfo.leading;

    WITH fPlotDoc DO
        BEGIN
            ( retrieve the length of our Axis )
            xScale := fxParam;
            yScale := fyParam;

            ( retrieve the coefficients )
            a := faParam;
            b := fbParam;
            c := fcParam;
            step := fStepParam;
        END;

    ClipRect(picRect);

    GetForeColor(oldFore);
    GetBackColor(oldBack);

    PenNormal;

    (Draw Graph Boundry)
    PicComment(picGrpBeg, 0, NIL);

    newColor := fPlotDoc.fPlotSpecs.theBackColor;
    RGBBackColor(newColor);

    PicComment(SetLineWidth,
        GetHandleSize(Handle(Width)), Handle(Width));
    FillRect(picRect, white);
    FrameRect(picRect);

    (Two Axis)
    newColor := fPlotDoc.fPlotSpecs.theAxisColor;
    RGBForeColor(newColor);

    ( find height of our View we will draw into )
    PictHorz := picRect.right - picRect.left;
    PictVert := picRect.bottom - picRect.top;

    ( find the center coordinates of the view )
    CenterHorz := PictHorz DIV 2;
    CenterVert := PictVert DIV 2;

    PicComment(picGrpBeg, 0, NIL);
    moveto(0, CenterVert);
    line(PictHorz, 0);
    moveto(CenterHorz, 0);
    line(0, PictVert);
    PicComment(picGrpEnd, 0, NIL);

    newColor := fPlotDoc.fPlotSpecs.theGraphColor;
    RGBForeColor(newColor);

    (Axis Text)
    ( Place -X axis label to the lowerleft of
      the axis line )
    moveto(AxisLabelIndent, CenterVert + leading);
    NumToString(-xScale, axisLabel);
    DrawString(axisLabel);

    ( Place +Y axis label to the topleft of the
      axis line )
    NumToString(yScale, axisLabel);
    moveto(CenterHorz - (StringWidth(axisLabel))

```

```

        + AxisLabelIndent), leading);
DrawString(axisLabel);

( Place +X axis label to the lowerright
  of the axis line )
NumToString(xScale,axisLabel);
moveto(PictHorz - (StringWidth(axisLabel)
  + AxisLabelIndent), CenterVert + leading);
DrawString(axisLabel);

( Place -Y axis label to the lowerleft
  of the axis line )
NumToString(-yScale,axisLabel);
moveto(CenterHorz - (StringWidth(axisLabel)
  + AxisLabelIndent), PictVert - leading);
DrawString(axisLabel);

(The Plot)
( calculate a scale factor for the
  PICT's axis in the view )
xPictScale := PictHorz / xScale;
yPictScale := PictVert / yScale;

( calculate our starting x,y point )
halfXScale := xScale / 2;
x := -halfXScale;
y := a * x * x + (b * x) + c;

( Scale it into the PICT's
  (and PostScript's) view space )
thisPt.h :=
  integer(round(x * xPictScale))+CenterHorz;
thisPt.v :=
  integer(round(-y * yPictScale))+CenterVert;

( Find where to stop and start drawing.
  LOOP until we reach all x values:
  1. Find our first visible line segment
  2. Plot until it disappears
  3. Find a suitable end point
  )

WHILE (NOT PtInRect(thisPt,picRect))
  & (x <= halfXScale) DO
  BEGIN
    x := x + step;
    y := a * x * x + (b * x) + c;
    thisPt.h :=
      integer(round(x * xPictScale))+CenterHorz;
    thisPt.v :=
      integer(round(-y * yPictScale))+CenterVert;
  END;

moveTo(thisPt.h,thisPt.v);
drawLine := TRUE;

PicComment(picGrpBeg, 0, NIL);

REPEAT
  x := x + step;
  y := (a * x * x) + (b * x) + c;
  thisPt.h :=
    integer(round(x*xPictScale))+CenterHorz;
  thisPt.v :=
    integer(round(-y*yPictScale))+CenterVert;
  IF NOT PtInRect(thisPt,picRect) THEN
    drawLine := FALSE
  ELSE
    BEGIN
      BEGIN
        IF drawLine THEN
          LineTo(thisPt.h,thisPt.v)
        ELSE
          BEGIN

```

```

        drawLine := TRUE;
        moveTo(thisPt.h,thisPt.v)
      END
    END
  UNTIL x >= halfXScale;

  PicComment(picGrpEnd, 0, NIL);

  RGBForeColor(oldFore);
  RGBBackColor(oldBack);

  PicComment(PicGrpEnd, 0, NIL); (of select all objects)

END;

(-----)
PROCEDURE TPlotView.Resize(width,
  height: VCoordinate; invalidate: BOOLEAN);
  OVERRIDE;
BEGIN
  invalidate := TRUE;
  INHERITED Resize(width, height, invalidate);
  ForceRedraw;
END;

(-----)
( Adds the text box containing the Quadratic equation solution
  to our PICT. )
PROCEDURE TSolutionView.AddToPict(picRect: Rect);
  CONST
    k1stLine = 1; k2ndLine = 2; k3rdLine = 3;
    k4thLine = 4; k5thLine = 5;

    kLeftJust = 1;
    kIndent = 2;

  TYPE
    Widpt = Point;
    WidPtr = ^WidPt;
    WidHdl = ^WidPtr;

    TTxtPicRec = PACKED RECORD
      tJus : Byte;
      tFlip : Byte;
      tRot : Integer;
      tLine : Byte;
      tCmnt : Byte;
    END;

  VAR
    TextClipRgn : RgnHandle;
    TxtPicRec : TTTxtPicRec;
    TxtPicPtr : QDPtr;
    TxtPicHdl : QDHandle;
    Width : WidHdl;

    clipBox : Rect;
    boxTop : INTEGER;
    boxLeft : INTEGER;
    leading : INTEGER;
    fInfo : FontInfo;
    LineNo : INTEGER;

    a,b,c : Real;
    x1,x2 : Real;
    z1,z2 : Real;

    newColor : RGBColor;

  BEGIN
    WITH fPlotDoc DO
      BEGIN
        ( retrieve the Solution )
        x1 := f1stRoot;
        x2 := f2ndRoot;

```

```

( retrieve the coefficients )
a := faParam;
b := fbParam;
c := fcParam;
END;

Width := Width1(NewHandle(sizeof(width)));
Width^.h := 10;
Width^.v := 1;

GetFontInfo(fInfo);
leading := fInfo.descent + fInfo.ascent + fInfo.leading;

PicComment(picGrpBeg, 0, NIL);

(Box )
PicComment(picGrpBeg, 0, NIL);
PicComment(SetLineWidth,
  GetHandleSize(Handle(Width)),
  Handle(Width));
fillRect(picRect, white);
frameRect(picRect);
PicComment(picGrpEnd, 0, NIL); (of box)

TextClipRgn := NewRgn;
FailNil(TextClipRgn);

GetClip(TextClipRgn);
clipBox := picRect;
ClipRect(clipBox);

newColor := fPlotDoc.fPlotSpecs.theTextColor;
RGBForeColor(newColor);

(Box Text)
( setup our Text Comment Handle )
TxtPicPtr := @TxtPicRec;
TxtPicHdl := @TxtPicPtr;
TxtPicRec.tJus := kLeftJust;
TxtPicRec.tFlip := 0; (no flip)
TxtPicRec.tRot := 0; (no rotation)
TxtPicRec.tLine := 2; (1 1/2 spacing)

(put the Comment into our PICT)
PicComment(TextBegin, sizeof(TTxtPicRec),
Handle(TxtPicHdl));

( Add the BOX text to the PICT )
boxTop := picRect.top;
boxLeft := picRect.left;

MoveTo(boxLeft + kIndent,
  boxTop + (k1stLine * leading));
DrawString(CONCAT('y=ax^2 + bx + c', chr(13)));

MoveTo(boxLeft + kIndent,
  boxTop + (k2ndLine * leading));
DrawString(CONCAT('a=', Real2Str(a,1), ', b=',
Real2Str(b,1), ', c=', Real2Str(c,1), chr(13)));

( The solution text is next )
MoveTo(boxLeft + kIndent,
  boxTop + (k3rdLine * leading));
DrawString(CONCAT('x1=', Real2Str(x1,3),
  ', x2=', Real2Str(x2,3), chr(13)));

( Display kind of Quadratic we have )
MoveTo(boxLeft + kIndent,
  boxTop + (k4thLine * leading));
CASE fPlotDoc.fRootType OF
  RealRoots :
    DrawString(CONCAT('Two Real Roots, x1, x2',
      chr(13)));
  SingleSolution :

```

```

    DrawString(CONCAT('Double Root', chr(13)));
  ComplexRoot :
    DrawString(CONCAT('Two Complex Roots ',chr(13)));
  OTHERWISE
    ;
END;

( Calculate the slope and add the
  string it to the PICT)
WITH fPlotDoc DO
  BEGIN
    z1 := -b / (2 * a);
    z2 := (4 * a * c - (b * b)) / (4 * b);
  END;

  MoveTo(boxLeft + kIndent,
    boxTop + (k5thLine * leading));
  DrawString(CONCAT('Slope 0 is (',
    Real2Str(z1,1), ', ', Real2Str(z2,1), ')',
    chr(13)));

  ( All done with our text )
  PicComment(TextEnd, 0, NIL);

  PicComment(picGrpEnd, 0, NIL); (of Box & text)

  DisposHandle(Handle(width));
  DisposeRgn(TextClipRgn);
END;

(-----)
PROCEDURE TPStyleCmd.DoIt;
  BEGIN
    fPlotDialog.fPlotDoc.fPlotSpecs := fNewPlotSpecs;
  END;

(-----)
PROCEDURE TPStyleCmd.ReDoIt;
  BEGIN
    fPlotDialog.fPlotDoc.fPlotSpecs := fNewPlotSpecs;
  END;

(-----)
PROCEDURE TPStyleCmd.UnDoIt;
  BEGIN
    fPlotDialog.fPlotDoc.fPlotSpecs := fOldPlotSpecs;
  END;

(-----)
PROCEDURE TPStyleCmd.IPStyleCmd(itsPlotDialog: TPlotDialog;
  itsNewStyle: PlotSpecsPtr; itsCmdNumber: CmdNumber);
  BEGIN
    ICommand(itsCmdNumber, itsPlotDialog.fPlotDoc,
      itsPlotDialog, NIL);

    fPlotDialog := itsPlotDialog;
    fOldPlotSpecs:=fPlotDialog.fPlotDoc.fPlotSpecs;
    fNewPlotSpecs:=itsNewStyle;
  END;

END.

WPlot.p Main Program FILE
(oooooooooooooooooooooooooooooooooooooooooooooooooooo)
Program Plot;
USES
  ( o MacApp )
  UMacApp,

  ( o Building Blocks )
  UDialog, UPrinting,

  ( o Implementation Use )
  SANE, ToolUtils, Fonts, Resources,

```

```

Script, PickerIntf, Packages,

( o the PlotUNIT )
UPlot;

VAR
  gPlotApplication: TPlotApplication;
  (The application object:)

(-----)
( THE MAIN PROGRAM )
BEGIN
  InitUMacApp(8);
    (Initialize the Toolbox,
    making 8 calls to MoreMasters:)
  InitPrinting;
    (Initialize the UPrinting unit:)

  NEW(gPlotApplication);
    (Allocate a new TPlotApplication object:)

  FailNIL(gPlotApplication);
  gPlotApplication.IPlotApplication(kFileType);
    (Initialize that new object:)

  gPlotApplication.Run;
    (Run the application.
    When it's done, exit.)

END.

```

PLOT.r-Plot resource FILE

```

/* Copyright © 1988-1989 Apple Computer, Inc.
   All rights reserved. */
/* Copyright © 1989 Software Architects, Inc.
   All rights reserved. */

```

```

#if qDebug
include "Debug.rsrc";
#endif
include "MacApp.rsrc";
include "Dialog.rsrc";
include "Printing.rsrc";

```

```
include "Plot" 'CODE';
```

```

#define kPlotView      1010
#define kViewRsrcID    1005

```

```

resource 'seg!' (256, purgeable) (
(
  "GOpen";
  "GClose";
  "GNonRes";
  "GSelCommand"
)
);

```

```

resource 'SIZE' (-1) (
  dontSaveScreen,
  acceptSuspendResumeEvents,
  enableOptionSwitch,
  canBackground,
  MultiFinderAware,
  backgroundAndForeground,
  dontGetFrontClicks,
  ignoreChildDiedEvents,
  is32BitCompatible,
  reserved,
  reserved,
  reserved,
  reserved,
  reserved,

```

```

  reserved,
  reserved,
  #if qDebug
    368 * 1024, /* ??? BALLPARK GUESSES ?????*/
    320 * 1024
  #else
    (300-32) * 1024, /* ??? BALLPARK GUESSES ?? */
    (200-32) * 1024
  #endif
);

resource 'DITL' (201, purgeable) (
(
  /* [1] */
  (160, 182, 180, 262), Button ( enabled, "OK" );
  /* [2] */
  (10, 75, 150, 316), StaticText ( disabled,
    "This is an implementation of the MacTutor Plotter Program."
    "\n\n written "
    "with MacApp© © 1985-1989 Apple Computer, Inc.");
  /* [3] */ (10, 20, 42, 52), Icon ( disabled, 1 )
)
);

resource 'ALRT' (1000, purgeable) (
  (44, 48, 130, 358),
  1000,
  (
    OK, visible, sound1,
    OK, visible, sound1,
    OK, visible, sound1,
    OK, visible, sound1
  )
);

resource 'ALRT' (201, purgeable) (
  (90, 100, 280, 412),
  201,
  (
    OK, visible, silent;
    OK, visible, silent;
    OK, visible, silent;
    OK, visible, silent
  )
);

resource 'cmnu' (1) (
  1,
  textMenuProc,
  0x7FFFFFFD,
  enabled,
  apple,
  (
    /* [1] */
    "About Plotter...", noIcon, noKey, noMark, plain, cAboutApp;
    /* [2] */
    "-", noIcon, noKey, noMark, plain, nocommand
  )
);

resource 'cmnu' (2) (
  2,
  textMenuProc,
  0x7FFFEFB,
  enabled,
  "File",
  (
    "New", noIcon, "N", noMark, plain, cNew;
    "Open...", noIcon, "O", noMark, plain, cOpen;
    "-", noIcon, noKey, noMark, plain, nocommand;
    "Close", noIcon, noKey, noMark, plain, cClose;
    "Save", noIcon, "S", noMark, plain, cSave;
    "Save As...", noIcon, noKey, noMark, plain, cSaveAs;
    "Save a Copy In...",

```

```

        noIcon, noKey, noMark, plain, cSaveCopy;
"Revert", noIcon, noKey, noMark, plain, cRevert;
"--", noIcon, noKey, noMark, plain,
nocommand;
"Page Setup...",
    noIcon, noKey, noMark, plain, cPageSetup;
"Print...", noIcon, noKey, noMark, plain, cPrint;
"--", noIcon, noKey, noMark, plain, nocommand;
"Quit", noIcon, "Q", noMark, plain, cQuit
    )
};

resource 'cmnu' (3) (
    3,
    textMenuProc,
    0x7FFFFBD,
    enabled,
    "Edit",
    (
        "Undo", noIcon, "Z", noMark, plain, cUndo;
        "--", noIcon, noKey, noMark, plain, nocommand;
        "Cut", noIcon, "X", noMark, plain, cCut;
        "Copy", noIcon, "C", noMark, plain, cCopy;
        "Paste",
            noIcon, "V", noMark, plain, cPaste;
        "Clear",
            noIcon, noKey, noMark, plain, cClear;
        "--", noIcon, noKey, noMark, plain, nocommand;
        "Show Clipboard",
            noIcon, noKey, noMark, plain, cShowClipboard
    )
);

/* Hierarchical menu with Style, Size & Font submenus */
resource 'cmnu' (7) (
    7,
    textMenuProc,
    allEnabled,
    enabled,
    "MacAppPlot",
    (
        "Plot", noIcon, noKey, noMark, plain, 1401;
        "--", noIcon, noKey, noMark, plain, nocommand;
        "Style", noIcon, "\0x1B", "\0x08", plain, 2401;
        "Size", noIcon, "\0x1B", "\0x09", plain, 2402;
        "Font", noIcon, "\0x1B", "\0x0A", plain, 2403;
        "Color", noIcon, "\0x1B", "\0x0B", plain, 2404
    )
);

/* Hierarchical Style Sub-menu */
resource 'cmnu' (8) (
    8,
    textMenuProc,
    allEnabled,
    enabled,
    "Style",
    (
        "Plain Text", noIcon, "P", noMark, plain, 2001;
        "Bold", noIcon, "B", noMark, bold, 2002;
        "Italic", noIcon, "I", noMark, italic, 2003;
        "Underline", noIcon, "U", noMark, underline, 2004;
        "Outline", noIcon, "O", noMark, outline, 2005;
        "Shadow", noIcon, "S", noMark, shadow, 2006;
        "Condensed", noIcon, noKey, noMark, condense, 2007;
        "Extended", noIcon, noKey, noMark, extend, 2008;
        "--", noIcon, noKey, noMark, plain, nocommand;
        "Left justified", noIcon, noKey, noMark, plain, 2201;
        "Center justified", noIcon, noKey, noMark, plain, 2202;
        "Right justified", noIcon, noKey, noMark, plain, 2203
    )
);

```

```

/* Hierarchical Size Sub-menu */
resource 'cmnu' (9) (
    9,
    textMenuProc,
    allEnabled,
    enabled,
    "Size",
    (
        "9 Point", noIcon, noKey, noMark, plain, 2109;
        "10 Point", noIcon, noKey, noMark, plain, 2110;
        "12 Point", noIcon, noKey, noMark, plain, 2112;
        "14 Point", noIcon, noKey, noMark, plain, 2114;
        "18 Point", noIcon, noKey, noMark, plain, 2118;
        "24 Point", noIcon, noKey, noMark, plain, 2124;
        "--", noIcon, noKey, noMark, plain, nocommand;
        "Grow selection", noIcon, "I", noMark, plain, 1198;
        "Shrink selection", noIcon, "I", noMark, plain, 1199
    )
);

/* Font menu for hierarchical or non-hierarchical system */
resource 'MENU' (10) (
    10,
    textMenuProc,
    allEnabled,
    enabled,
    "Font",
    (
    )
);

resource 'cmnu' (11) (
    11,
    textMenuProc,
    allEnabled,
    enabled,
    "Color",
    (
        "Set Graph Color...", noIcon, noKey, noMark, plain, 2503;
        "Set Axis Color...", noIcon, noKey, noMark, plain, 2504;
        "Set Text Color...", noIcon, noKey, noMark, plain, 2501;
        "Set Background Color...", noIcon, noKey, noMark, plain, 2502
    )
);

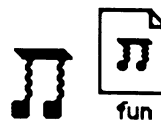
resource 'cmnu' (128) (
    128,
    textMenuProc,
    allEnabled,
    enabled,
    "Buzzwords",
    (
        "Style Change",
            noIcon, noKey, noMark, plain, cStyleChange;
        "Size Change",
            noIcon, noKey, noMark, plain, 2100;
        "Justification Change",
            noIcon, noKey, noMark, plain, 2200;
        "Font Change",
            noIcon, noKey, noMark, plain, 2300;
        "Color Change",
            noIcon, noKey, noMark, plain, 2500;
        "Page Setup Change",
            noIcon, noKey, noMark, plain, cChangePrinterStyle
    )
);

/* Displayed menus on a non-hierarchical system */
resource 'MBAR' (128) (
    (1; 2; 3; 8; 9; 10; 11)
);

```


Intermediate Mac'ing

'snd ' Zoundz Great!



Zoundz 1.0

Kirk Chase
Anaheim, CA
Volume 5, Number 9

Piano Lessons—Yech!

When I was a boy, my mother made me take piano lessons (a fate that has fallen upon many young boys). I wasn't too interested in the piano, so I soon traded the ivory keyboard for the Qwerty keyboard. I can still hear my mother telling me that I'll regret the day I stopped playing the piano; I didn't think so—until now.

I wanted to add sound to an application I was writing. Not knowing anything about sound other than the simple beep, I decided the Macintosh 'snd ' resource was my next attack of study. So after an evening with Inside Macintosh, Vol. 5, I decided I had to be a MIDI maniac to understand a lot of the information found there, but I did manage to get a grasp on some of the simpler elements. So here is a simple overview.

To make a sound, you basically create a sound channel to a particular synthesizer and pass it a list of sound commands which, among other things, generate sounds—simple. A synthesizer allows music to be played in a certain way; there is currently a note synthesizer, a wave synthesizer, a sampled sound synthesizer, and some MIDI synthesizers. Which one you use depends on how you are going to generate the sound. I chose the note synthesizer for my sound generating application.

A sound channel is a record that holds information for processing the sound. It contains modifiers, call-back procedures, and a queue of sound commands. The sound commands are in a 'snd ' resource format.

The 'snd ' Resource

The 'snd ' resource has currently two types. The second type is used for representing an instrument or digitally recorded sound. Type one is used by us non-MIDI people. The first word in the 'snd ' resource specifies this format type, 1 or 2. I use format type 1.

The second word tells how many synthesizers and modifiers that follow. A modifier, for a quick explanation, is some code that alters the sound generated in some manner. I use this second word to open up a note synthesizer. This word is followed by a word identifying the synthesizer and a long integer giving the initialization procedure, if any. This is repeated for each synthesizer and modifier as specified in the second word.

After that, there is a word telling how many sound commands are to follow. Then the sound commands and any needed data are given. The format of the sound command is a word specifying which sound command, another word for param1, and a long for param2. If this is all confusing, see figure 1.

Sound Commands

For my application, I use the simple note synthesizer. Out

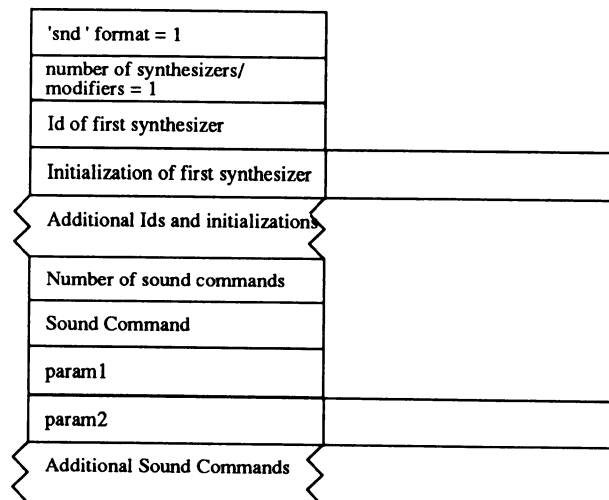


Figure 1. 'snd ' Resource Format

of all the commands that generate sound, the note synthesizer recognizes only the note, rest, quiet, frequency, amplitude, and timbre commands. I use only the note, timbre, rest, and quiet commands in my application.

The note command has the ability to specify the pitch, amplitude, and duration. To further confuse matters, the pitch may be specified in two ways. The first way is to use a value between 0 and 127. This is then converted to a piano key (60 is middle C; 61 is C#). The other way is to give the actual frequency. I use the first way. The amplitude (0...255) is also added to the pitch and stored in the long (See figure 2). The duration, param1, is specified in milliseconds.

AA = Amplitude (0...255). Put in the high order byte of a long integer.
PP = Pitch (0...127). Put in low order byte of a long integer.
Add two long integers together and assign to param2 of sound command

```
AA000000
+000000PP
=AA0000PP
```

Figure 2. Combining Pitch and Amplitude

Zoundz 1.0

Zoundz gives the user the ability to draw a wave type description of the sound. There are four graphs the user draws to

generate the sound—frequency, amplitude, duration, and timbre. Each vertical column is a note. You may also control the display of each graph (the drawing goes faster if you turn off the drawing of the other graphs).

The user may specify the exact values of a particular note. Use the scroll bar below the graphs to move the note you wish to modify to the left edge of the rectangle. The individual values may then be adjusted using the scroll bars in the upper right corner of the window. You may also specify which note to start and end your selection. When you are ready, just press the “Play Sound” button (be sure to specify the selection range). Figure 3 shows the Zoundz window.

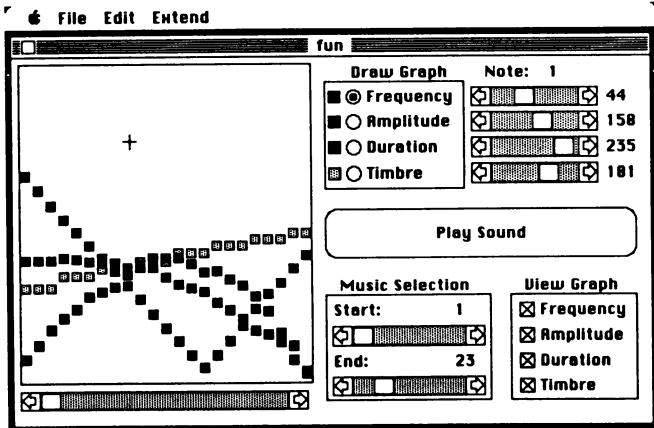


Figure 3. Zoundz Window

Some editing shortcuts can be found under the “Extend” menu. Just set the selection range and select one of the menu items under this menu. The value selected in the menu of the current note (left edge) is given to the notes in the selection range.

When you save the sound, the values for each of the 100 possible notes are saved in the data fork. In the resource fork, a ‘snd’ resource is created with your sound. The name is the same as the one you gave to the file, and its ID=9000. Just use ResEdit to paste the sound into the system file (you may wish to give it a new ID number). You may also get a printed dump of your ‘snd’ resource by selecting the print option from the “File” menu.

Zoundz is pretty nice. It handles Finder startups for the documents to either print or open. It gives us non-musicians the chance to sketch our sound. Zoundz is not very good at producing composed music. I did however get it to play “Happy Birthday”. Try drawing patterns and listen to the outcome. You might create something that zoundz good.

Listing: MyGlobals

```
unit MyGlobals;
interface
uses
  PrintTraps, Sound;
const
  L_Apple = 1001;      (Menu list)
  MI_About_Zoundz = 1;
  L_File = 1002;      (Menu list)
  MI_New = 1;
  MI_Open = 2;
```

Debug Windows

Zoundz.tif		
Options	File (by build order)	Size
	Runtime.lib	18258
	Interface.lib	8104
	PrintCalls.lib	1096
D N V R	PrintTraps.p	0
D N V R	Sound.p	0
D N V R	MyGlobals.Pas	0
D N V R	MySound.Pas	434
D N V R	InitTheMenus.Pas	98
D N V R	Message.Pas	230
D N V R	save_changes.Pas	210
D N V R	MyFileStuff.Pas	1694
D N V R	MyPrintStuff.Pas	3054
D N V R	About.Pas	294
D N V R	Untitled.Pas	4792
D N V R	HandleTheMenus.Pas	1138
D N V R	Zoundz.Pas	1674

Figure 4. Zoundz Build Order

```
MI_Close = 4;
MI_Save = 5;
MI_Save_As = 6;
MI_Page_Setup = 8;
MI_Print = 9;
MI_Quit = 11;
L_Edit = 1003;      (Menu list)
MI_Undo = 1;
MI_Cut = 3;
MI_Copy = 4;
MI_Paste = 5;
L_Extend = 1004;    (Menu list)
MI_Frequency = 1;
MI_Amplitude = 2;
MI_Duration = 3;
MI_Timbre = 4;
I_Yes = 1;
I_Cancel = 2;
I_No = 3;

type
  DocPtr = ^DocRec;
  DocRec = record (Sound Doc Structure)
    Freq, Amp, Dur, Timbre: array[1..100] of integer;
    EndValue, StartValue: integer;
  end;
  MySoundRec = packed record (Snd structure)
    format: integer;
    SynthCount: integer;
    SynthType: integer;
    SynthInit: longint;
    CommandCount: integer;
    MySounds: array[1..202] of SndCommand;
  end;
  MySoundPtr = ^MySoundRec;
  MySoundHdl = ^MySoundPtr;

var
  FreqText, AmpText, DurText, TimbreText: Str255;
  NoteText, StartText, EndText: Str255;
  NoteIndex, DrawTool: integer; (Indices)
  MyDoc: DocPtr; (Sound Document)
  Dirty: boolean;
  NoteRect, FreqRect, AmpRect, DurRect, TimbreRect: Rect;
  EndRect, StartRect, NotePalette: Rect;
  MyHandle: Handle; (Various Handles)
  MySoundHandle: MySoundHdl;
  AppleMenu: MenuHandle; (Menu handle)
  M_File: MenuHandle;
  M_Edit: MenuHandle;
  M_Extend: MenuHandle;
```

```

MyWindow: WindowPtr; (Window pointer)
FileName: str255; (File Stuff)
volRefNum, FileStatus: integer;
theSquare, theWatch: CursHandle; (Cursor Stuff)
ThePrintRec: THPrint; (Printing Stuff)
ThePrintPort: TPrPort;
PrintStatus: TPrStatus;
PageRect: rect;
implementation
end.

Listing: MySound.pas

unit MySound;
interface
uses
    PrintTraps, Sound, MyGlobals;
    (creates a 'snd' resource)
procedure CreateSndResource (StartSel, EndSel: integer);
implementation
procedure CreateSndResource;
    var
        i, j: integer;
        amplong, freqlong: longint;
        lastTimbre: integer;
        theErr: OSErr;
        theSize: integer;
begin
    lastTimbre := -1;
    if Handle(MySoundHandle) <> nil then
        begin
            DisposHandle(Handle(MySoundHandle));
            DisposHandle(MyHandle);
        end;
    MySoundHandle :=
        MySoundHdl(NewHandle(sizeof(MySoundRec)));
    with MySoundHandle^^ do
        begin
            format := 1; (set up header stuff)
            SynthCount := 1;
            SynthType := 1;
            SynthInit := 0;
            with MyDoc^^ do
                begin
                    j := 0;
                    for i := StartSel to EndSel do
                        begin (get sound commands)
                            j := j + 1;
                            if timbre[i] <> lastTimbre then
                                begin
                                    MySounds[j].cmd := timbreCmd;
                                    MySounds[j].param1 := timbre[i];
                                    MySounds[j].param2 := 0;
                                    lastTimbre := timbre[i];
                                    j := j + 1;
                                end; (of timbre command)
                            if freq[i] = 128 then (is it a rest?)
                                begin
                                    MySounds[j].cmd := restCmd;
                                    MySounds[j].param1 := dur[i];
                                    MySounds[j].param2 := 0;
                                end
                            else (no, regular note)
                                begin
                                    amplong := amp[i];
                                    amplong := BitAnd(BitShift(amplong, 24),
$FF000000);
                                    freqlong := BitAnd(freq[i], $000000FF);
                                    MySounds[j].cmd := noteCmd;
                                    MySounds[j].param1 := dur[i];
                                    MySounds[j].param2 := amplong +
freqLong;

```

```

                                end;
                                end; (of for loop)
                            end; (with MyDoc)
                            j := j + 1;
                            MySounds[j].cmd := noteCmd; (turn off sound)
                            MySounds[j].param1 := 0;
                            MySounds[j].param2 := 0;
                            j := j + 1;
                            MySounds[j].cmd := quietCmd;
                            MySounds[j].param1 := 0;
                            MySounds[j].param2 := 0;
                            CommandCount := j;
                        end; (of with MySoundHandle)
                        theSize := 12 + (j * 8);
                        MyHandle := Handle(MySoundHandle);
                        theErr := HandToHand(MyHandle);
                        SetHandleSize(MyHandle, Size(theSize));
                    end; (of CreateSndResource)
end.

```

Listing: InitTheMenus.pas

```

unit InitTheMenus;
interface
uses
    PrintTraps, Sound, MyGlobals;
procedure Init_My_Menus; (Initialize the menus)
implementation
procedure Init_My_Menus; (Initialize the menus)
    const
        Menu1 = 1001; (Menu resource ID)
        Menu2 = 1002; (Menu resource ID)
        Menu3 = 1003; (Menu resource ID)
        Menu4 = 1004;
    begin
        (Start of Init_My_Menus)
        ClearMenuBar; (Clear any old menu bars)
        AppleMenu := GetMenu(Menu1);
        InsertMenu(AppleMenu, 0);
        AddResMenu(AppleMenu, 'DRVr');
        M_File := GetMenu(Menu2);
        InsertMenu(M_File, 0);
        M_Edit := GetMenu(Menu3);
        InsertMenu(M_Edit, 0);
        M_Extend := GetMenu(Menu4);
        InsertMenu(M_Extend, 0);
        DrawMenuBar;
    end; (End of procedure Init_My_Menus)
end.

```

Listing: Message.pas

```

unit Message;
interface
procedure A_Message (s0, s1, s2, s3: str255; var theItem:
integer);
implementation
procedure A_Message;
    var
        itemHit: Integer;
        AlertResHandle: AlertHndl;
        tempRect: Rect;
    begin
        (Start of alert handler)
        ParamText(s0, s1, s2, s3);
        AlertResHandle := AlertHndl(GetResource('ALRT', 4));
        HLock(Handle(AlertResHandle));
        tempRect := AlertResHandle^^.boundsRect;
        tempRect.Left := ((screenBits.Bounds.Right -
screenBits.Bounds.Left) - (tempRect.Right - tempRect.Left))
div 2; (Center Horz)
        tempRect.Bottom := tempRect.Top +
(AlertResHandle^^.boundsRect.Bottom -
AlertResHandle^^.boundsRect.Top);
        tempRect.Right := tempRect.Left +
(AlertResHandle^^.boundsRect.Right -
AlertResHandle^^.boundsRect.Left);
        AlertResHandle^^.boundsRect := tempRect;

```

```

    theItem := NoteAlert(4, nil);
    HUnlock(Handle(AlertResHandle));
end;
    (End of procedure)
end.
    (End of unit)

```

Listing: save_changes.pas

```

unit save_changes;
interface
    function D_save_changes: integer;
implementation
    const
        I_Yes = 1;
        I_Cancel = 2;
        I_No = 3;
        I_x = 4;
    var
        ExitDialog: boolean;
        DoubleClick: boolean;
        MyPt: Point;
        MyErr: OSErr;
    function D_save_changes;
    var
        GetSelection: DialogPtr;
        tempRect: Rect;
        DType: Integer;
        Ditem: Handle;
        itemHit: Integer;
    procedure Refresh_Dialog;
    var
        rTempRect: Rect;
    begin
        SetPort(GetSelection); (Point to our dialog window)
        GetDitem(GetSelection, I_Yes, DType, Ditem, tempRect);
        PenSize(3, 3);
        InsetRect(tempRect, -4, -4);
        FrameRoundRect(tempRect, 16, 16);
        PenSize(1, 1);
    end;
    begin
        (Start of dialog handler)
        GetSelection := GetNewDialog(3, nil, Pointer(-1));
        ShowWindow(GetSelection);
        SelectWindow(GetSelection);
        SetPort(GetSelection);
        Refresh_Dialog;
        ExitDialog := FALSE;
    repeat
        ModalDialog(nil, itemHit);
        D_save_changes := itemHit;
        if (ItemHit = I_Yes) or (ItemHit = I_Cancel) or
        (ItemHit = I_No) then
            ExitDialog := TRUE;
        until ExitDialog;
        DisposDialog(GetSelection);
    end;
end.
    (End of unit)

```

Listing: MyFileStuff.pas

```

unit MyFileStuff;
interface
    uses
        PrintTraps, Sound, MyGlobals, Message, MySound;
    procedure doSave;
    procedure doSaveAs;
    procedure OpenFile;
    procedure doOpen;
implementation
    const
        SFPutLeft = 82;
        SFPutTop = 50;
        myType = 'ZZDC';
        myCreator = 'KCZZ';
    var
        SFPutPt: Point;
        theReply: SFRReply;

```

```

refNum, resRef: integer;
bytes: longint;
title: str255;
theLength: longint;
theErr: OSErr;
theItem: integer;
oldHandle: Handle;

```

```

function HandleError (theStr: Str255; theError: OSErr; chk,
chksum: integer; CloseIt: boolean): boolean;
var
    STemp: str255;
begin
    if (theError <> noErr) then
        begin
            A_Message(theStr, '', '', '', theItem);
            if CloseIt then
                begin
                    theErr := FSClose(refNum);
                    theErr := FlushVol(nil, VolRefNum);
                end;
            HandleError := true;
        end
    else if (chk <> chksum) then
        begin
            A_Message(theStr, 'Checksum Error', '', '', theItem);
            if CloseIt then
                begin
                    theErr := FSClose(refNum);
                    theErr := FlushVol(nil, VolRefNum);
                end;
            HandleError := true;
        end
    else
        HandleError := false;
    end;

    procedure doSave;
    begin
        if VolRefNum = 0 then
            begin (bad volume reference number)
                A_Message('Bad Volume Number', '', '', '', theItem);
                Exit(doSave);
            end
        else (good volume reference number)
            begin
                theErr := FSOpen(FileName, VolRefNum, refNum);
                if HandleError('Could Not Open File', theErr, 0, 0,
                false) then
                    Exit(doSave)
                else (file was open ok)
                    begin
                        theErr := SetFPos(refNum, FSFromStart, 0);
                        if HandleError('Could Not Open File Position',
                        theErr, 0, 0, true) then
                            Exit(doSave)
                        else (ready to go)
                            begin
                                bytes := sizeof(DocRec);
                                theLength := bytes;
                                theErr := FSWrite(refNum, bytes,
                                ptr(MyDoc));
                                if HandleError('Trouble Writing To File',
                                theErr, bytes, theLength, true) then
                                    Exit(doSave);
                                end;
                            end;
                        (of file open ok)
                        theErr := FSClose(refNum);
                        theErr := FlushVol(nil, VolRefNum);
                        theErr := SetVol(nil, VolRefNum);
                        CreateResFile(FileName);
                        resRef := OpenResFile(FileName);
                        if resRef = -1 then
                            begin (could not be opened)

```

```

        A_Message('Could not write sound to', FileName,
'', '', theItem);
        Exit(doSave);
    end
    else (ready to write out sound)
    begin
        OldHandle := GetResource('snd ', 9000);
        if OldHandle <> nil then
            begin (existing sound to remove)
                RmveResource(OldHandle);
                DisposHandle(OldHandle);
                UpdateResFile(resRef);
            end;
        CreateSndResource(MyDoc^.StartValue,
MyDoc^.EndValue);
        AddResource(MyHandle, 'snd ', 9000, FileName);
        UpdateResFile(resRef);
        CloseResFile(resRef);
        DisposHandle(MyHandle);
        DisposHandle(Handle(MySoundHandle));
    end;
    end; (of good vol ref num)
end; (of doSave)

procedure doSaveAs;
begin
    SetPt(SFPutPt, SFPutLeft, SFPutTop);
    GetWTitle(MyWindow, title);
    SFPutFile(SFPutPt, 'Save Zoundz as...', title, nil, theRe-
ply);
    if theReply.good then
        begin
            theErr := Create(theReply.fname, theReply.vRefNum,
myCreator, myType);
            if theErr <> noErr then (duplicate or problem)
                begin
                    if theErr = dupFNerr then
                        begin (duplicate)
                            theErr := FSDelete(theReply.fname,
theReply.vRefNum);
                            if theErr <> noErr then
                                begin (cannot delete file)
                                    A_Message('Cannot Delete File.', '', '',
'', theItem);
                                    Exit(doSaveAs);
                                end;
                            theErr := Create(theReply.fname,
theReply.vRefNum, myCreator, myType);
                            if theErr <> noErr then
                                begin (error in creating after deleting
duplicate)
                                    A_Message('Cannot Create',
theReply.fname, '', '', theItem);
                                    Exit(doSaveAs);
                                end;
                            end
                        end
                    else (a problem)
                        begin
                            A_Message('Cannot Create', theReply.fname,
'', '', theItem);
                            Exit(doSaveAs);
                        end;
                    end (duplicate or problem)
                end
            else (ready to save)
                begin
                    VolRefNum := theReply.vRefNum;
                    FileName := theReply.fname;
                    SetWTitle(MyWindow, FileName);
                    theErr := FlushVol(nil, VolRefNum);
                    doSave;
                end; (ready to save)
            end; (good reply)
        end;
    end;
end;

```

```

procedure OpenFile;
begin
    theErr := FSOpen(FileName, VolRefNum, refNum);
    if HandleError('Could Not Open File', theErr, 0, 0, false)
then
        Exit(OpenFile)
    else (file was open ok)
        begin
            theErr := SetFPos(refNum, FSFromStart, 0);
            if HandleError('Could Not Open File Position',
theErr, 0, 0, true) then
                begin
                    VolRefNum := 0;
                    Exit(OpenFile);
                end
            else (ready to go)
                begin
                    bytes := sizeof(DocRec);
                    theLength := bytes;

                    theErr := FSRead(refNum, bytes, ptr(MyDoc));
                    if HandleError('Trouble Reading File', theErr,
bytes, theLength, true) then
                        begin
                            VolRefNum := 0;
                            Exit(OpenFile);
                        end;
                    end;
                end; (of file open ok)
            theErr := FSClose(refNum);
            theErr := FlushVol(nil, VolRefNum);
            MyWindow := nil;
            NoteText := '1';
            NumToString(MyDoc^.StartValue, StartText);
            NumToString(MyDoc^.EndValue, EndText);
            NumToString(MyDoc^.freq[1], FreqText);
            NumToString(MyDoc^.amp[1], AmpText);
            NumToString(MyDoc^.dur[1], DurText);
            NumToString(MyDoc^.timbre[1], TimbreText);
            NoteIndex := 1;
            DrawTool := 1;
            MySoundHandle := nil;
            MyHandle := nil;
        end;
    end;

procedure doOpen;
var
    theTypes: SFTypelist;
begin
    SetPt(SFPutPt, SFPutLeft, SFPutTop);
    theTypes[0] := myType;
    SFGetFile(SFPutPt, 'Open Zoundz file...', nil, 1, theTypes,
nil, theReply);
    VolRefNum := 0;
    if theReply.good then
        begin
            VolRefNum := theReply.vRefNum;
            FileName := theReply.fname;
            OpenFile;
        end;
    end;
end.

```

Listing: MyPrintStuff.pas

```

unit MyPrintStuff;
interface
    uses
        PrintTraps, Sound, MyGlobals, MySound, Message;
    procedure doSetUp;
    procedure doPrint;
implementation
    var
        theItem: integer;
    procedure doSetUp;

```

```

var
    confirmed: boolean;
begin
    PrOpen;
    InitCursor;
    confirmed := PrValidate(ThePrintRec);
    confirmed := PrStdDialog(ThePrintRec);
    if PrError <> noErr then
        A_Message('Problem with style dialog', '', '', '',
theItem)
    else
        PageRect := ThePrintRec^.prInfo.rpage;
        PrClose;
    end;

procedure PrintIt;
var
    leftEdge, lineTop, lineBottom, lineSize: integer;
    title: str255;
    i: integer;
procedure NumToHexString (n: longint; var s: str255);
var
    d, i: integer;
begin
    s := '';
    i := 32;
    while i > 0 do
        begin
            d := BitAnd(n, 15);
            n := BitShift(n, -4);
            i := i - 4;
            if d < 10 then
                s := concat(chr(ord('0') + d), s)
            else
                s := concat(chr(ord('A') + d - 10), s);
        end;
    end;

procedure LineFeed;
begin
    lineTop := lineTop + lineSize;
    lineBottom := lineBottom + lineSize;
    MoveTo(leftEdge, lineBottom);
end;

procedure PrintHeader;
var
    s1: str255;
begin
    s1 := 'Snd name is ';
    s1 := concat(s1, title, ' ');
    MoveTo(leftEdge, lineBottom);
    TextFace([bold]);
    DrawString(s1);
    TextFace([]);
    LineFeed;
    LineFeed;
end;

procedure PrintFirstPart;
var
    s1, s2: str255;
    num: longint;
begin
    num := MySoundHandle^.format;
    s1 := 'Snd Format = ';
    NumToString(num, s2);
    s1 := concat(s1, s2);
    DrawString(s1);
    LineFeed;
    num := MySoundHandle^.SynthCount;
    s1 := 'Synthizers = ';
    NumToString(num, s2);
    s1 := concat(s1, s2);

```

```

    DrawString(s1);
    LineFeed;
    num := MySoundHandle^.SynthType;
    s1 := 'Snd Format = ';
    NumToString(num, s2);
    s1 := concat(s1, s2, ' (noteSynth)');
    DrawString(s1);
    LineFeed;
    num := MySoundHandle^.SynthInit;
    s1 := 'Snd Initialization = ';
    NumToHexString(num, s2);
    s1 := concat(s1, '$', s2);
    DrawString(s1);
    LineFeed;
    num := MySoundHandle^.CommandCount;
    s1 := 'Number of Sound Commands = ';
    NumToString(num, s2);
    s1 := concat(s1, s2);
    DrawString(s1);
    LineFeed;
    DrawString(' # cmd      param1      param2      Description');
    MoveTo(leftEdge, lineBottom + 2);
    LineTo(PageRect.right, lineBottom + 2);
    MoveTo(leftEdge, lineBottom);
    LineFeed;
end;

procedure PrintNote (i: integer);
var
    s1, s2, s3: str255;
    num: longint;
    c, p1: integer;
    p2: longint;
begin
    c := MySoundHandle^.MySounds[i].cmd;
    p1 := MySoundHandle^.MySounds[i].param1;
    p2 := MySoundHandle^.MySounds[i].param2;
    num := i; (put index number)
    NumToString(num, s1);
    if i < 10 then
        s1 := concat(' ', s1);
    if i < 100 then
        s1 := concat(' ', s1);
    s1 := concat(s1, ' ');
    NumToString(c, s2);
    if c < 10 then
        s2 := concat(' ', s2);
    s1 := concat(s1, s2, ' $');
    NumToHexString(p1, s2);
    NumToHexString(p2, s3);
    s1 := concat(s1, s2, ' $', s3, ' ');
    case c of
        quietCmd:
            begin
                s1 := concat(s1, 'quietCmd - The End');
            end;
        timbreCmd:
            begin
                s1 := concat(s1, 'timbreCmd - Value ');
                NumToString(p1, s2);
                s1 := concat(s1, s2);
            end;
        restCmd:
            begin
                s1 := concat(s1, 'restCmd - Rest ');
                NumToString(p1, s2);
                s1 := concat(s1, s2, ' milliseconds');
            end;
        noteCmd:
            begin
                s1 := concat(s1, 'noteCmd - Note ');
                num := BitAnd(p2, $FF);
                NumToString(num, s2);

```

```

        s1 := concat(s1, s2, ' Amp. ');
        num := BitAnd(BitShift(p2, -24), $FF);
        NumToString(num, s2);
        s1 := concat(s1, s2, ' Duration ');
        NumToString(p1, s2);
        s1 := concat(s1, s2, ' milliseconds');
    end;
    otherwise
    begin
        s1 := concat(s1, 'Unknown sound command');
    end;
end;
DrawString(s1);
end;

begin
(set up position)
PenNormal;
TextFont(monaco);
TextFace(1);
TextSize(9);
lineTop := PageRect.top;
lineSize := 12;
lineBottom := lineTop + lineSize;
leftEdge := 30;
GetWTitle(MyWindow, title);
PrOpenPage(ThePrintPort, nil); (open page)
PrintHeader; (print header)
PrintFirstPart; (print first part)
for i := 1 to MySoundHandle^.CommandCount do
begin
    if lineBottom > PageRect.bottom then
    begin (if position is too great)
        PrClosePage(ThePrintPort); (close page)
        PrOpenPage(ThePrintPort, nil); (open page)
        lineTop := PageRect.top;
        lineBottom := lineTop + lineSize;
        PrintHeader; (print header)
    end;
    DrawString(' # cmd param1 param2 Description');
    MoveTo(leftEdge, lineBottom + 2);
    LineTo(PageRect.right, lineBottom + 2);
    MoveTo(leftEdge, lineBottom);
    LineFeed;
    end;
    PrintNote(i); (print note)
    LineFeed;
end;
PrClosePage(ThePrintPort); (close page)
end;

procedure doPrint;
var
    DoIt: boolean;
    myPrPort: TPrPort;
    savePort: GrafPtr;
    copies, count: integer;
begin
    GetPort(savePort);
    SetCursor(arrow);
    PrOpen;
    if PrError = noErr then
    begin
        DoIt := PrValidate(ThePrintRec);
        DoIt := PrJobDialog(ThePrintRec);
        if PrError <> noErr then
            A_Message('Problem with job dialog', '', '', '',
theItem);
        if DoIt then
        begin (print document)
            SetCursor(theWatch);
            ThePrintPort := PrOpenDoc(ThePrintRec, nil, nil);
            if PrError = noErr then
            begin (ok port)

```

```

                CreateSndResource(MyDoc^.StartValue,
MyDoc^.EndValue);
                copies := ThePrintRec^.prJob.iCopies;
                PageRect := ThePrintRec^.prInfo.rpage;
                for count := 1 to copies do
                begin (copies loop)
                    PrintIt; (print the document)
                end; (copies loop)
                DisposalHandle(MyHandle);
                DisposalHandle(Handle(MySoundHandle));
                MyHandle := nil;
                MySoundHandle := nil;
            end
        else (bad port)
            A_Message('Open Document Error', '', '', '',
theItem);
            PrCloseDoc(ThePrintPort);
            if (ThePrintRec^.prJob.bJDocLoop = bSpoolLoop)
and (PrError = noErr) then
                PrPicFile(ThePrintRec, nil, nil, nil,
PrintStatus);
            end; (printing document)
        end;
        PrClose;
        SetPort(savePort);
        SetCursor(arrow)
    end;
end.

```

Listing: About.pas

```

unit About;
interface
    procedure D_About;
implementation
    const
        I_OK = 1;
        I_x = 2;
        I_x3 = 3;
    var
        ExitDialog: boolean;
        DoubleClick: boolean;
        MyPt: Point;
        MyErr: OSErr;

    procedure D_About;
    var
        GetSelection: DialogPtr;
        tempRect: Rect;
        DType: Integer;
        DItem: Handle;
        itemHit: Integer;
    procedure Refresh_Dialog;
    var
        rTempRect: Rect;
    begin
        SetPort(GetSelection);
        GetDItem(GetSelection, I_OK, DType, DItem, tempRect);
        PenSize(3, 3);
        InsetRect(tempRect, -4, -4);
        FrameRoundRect(tempRect, 16, 16);
        PenSize(1, 1);
    end;
    begin
        GetSelection := GetNewDialog(2, nil, Pointer(-1));
        tempRect := GetSelection^.portRect;
        tempRect.Top := ((screenBits.Bounds.Bottom -
screenBits.Bounds.Top) - (tempRect.Bottom - tempRect.Top)) div
2;
        tempRect.Left := ((screenBits.Bounds.Right -
screenBits.Bounds.Left) - (tempRect.Right - tempRect.Left))
div 2;
        MoveWindow(GetSelection, tempRect.Left, tempRect.Top,
TRUE);
        ShowWindow(GetSelection);
    end;
end.

```

```

SelectWindow(GetSelection);
SetPort(GetSelection);
Refresh_Dialog;
ExitDialog := FALSE;

repeat
  ModalDialog(nil, itemHit);
  GetItem(GetSelection, itemHit, DType, Ditem, tempRect);
  if (ItemHit = I_OK) then
    begin
      ExitDialog := TRUE;
    end;
until ExitDialog;
DisposDialog(GetSelection);
end;
end. (End of unit)

```

Listing: Untitled.pas

```

unit Untitled;
interface
uses
  PrintTraps, Sound, MyGlobals, MySound;
  (Initialize us so all our routines can be activated)
procedure Init_Untitled;
  (Close our window)
procedure Close_Untitled (whichWindow: WindowPtr);
  (Open our window and draw everything)
procedure Open_Untitled;
  (Update our window, someone uncovered a part of us)
procedure Update_Untitled (whichWindow: WindowPtr);
  (Handle action to our window, like controls)
procedure Do_Untitled (myEvent: EventRecord);
implementation
const
  B_Play_Sound = 26;      (Button ID)
  CB_Timbre = 16;         (Checkbox IDs)
  CB_Duration = 15;
  CB_Amplitude = 14;
  CB_Frequency = 13;
  RB_Timbre = 20;         (Radio IDs)
  RB_Duration = 19;
  RB_Amplitude = 18;
  RB_Frequency = 17;
  I_DurationScrollbar = 41; (Scroll bar IDs)
  I_AmplitudeScrollbar = 40;
  I_FrequencyScrollbar = 39;
  I_TimbreScrollbar = 38;
  I_EndScrollbar = 31;
  I_StartScrollbar = 27;
  I_NoteScrollbar = 12;
var
  tempRect: Rect; (Temporary rectangle)
  sTemp: Str255;   (Get text entered, temp holding)
  C_EndScrollbar: ControlHandle;
  C_StartScrollbar: ControlHandle;
  C_NoteScrollbar: ControlHandle;
  R1Control: array[1..4] of ControlHandle;
  C_Play_Sound: ControlHandle;
  C_Timbre: ControlHandle;
  C_Duration: ControlHandle;
  C_Amplitude: ControlHandle;
  C_Frequency: ControlHandle;
  C_DurationScrollbar: ControlHandle;
  C_AmplitudeScrollbar: ControlHandle;
  C_FrequencyScrollbar: ControlHandle;
  C_TimbreScrollbar: ControlHandle;

  (=====)
  (Initialize us so all our routines can be activated)
  procedure Init_Untitled;
  var
    i: integer;
  begin
    (Start of Window initialize routine)
    MyWindow := nil;

```

```

    NoteText := '1'; (Init Texts)
    StartText := '1';
    EndText := '1';
    FreqText := '0';
    AmpText := '0';
    DurText := '0';
    TimbreText := '0';
    NoteIndex := 1; (Init Miscellaneous)
    DrawTool := 1;
    MyDoc^.EndValue := 1;
    MyDoc^.StartValue := 1;
    MySoundHandle := nil;
    MyHandle := nil;
    volRefNum := 0;
    for i := 1 to 100 do (initialize arrays)
      begin
        MyDoc^.freq[i] := 0;
        MyDoc^.amp[i] := 0;
        MyDoc^.dur[i] := 0;
        MyDoc^.timbre[i] := 0;
      end;
    end; (End of Init_Untitled)

  (=====)
  (Close our window)
  procedure Close_Untitled;
  begin
    if (MyWindow <> nil) and ((MyWindow = whichWindow) or
      (ord4(whichWindow) = -1)) then
      begin
        DisposeWindow(MyWindow);
        MyWindow := nil;
      end;
    end; (End of Close_Untitled)

  (=====)
  (draws a square according to the pattern)
  procedure DrawSquare (vert, horiz: integer; thePat:
    pattern);
  var
    theSquare: rect;
  begin
    SetRect(theSquare, horiz, vert, horiz + 10, vert + 10);
    InSetRect(theSquare, 1, 1);
    PenNormal;
    FillRect(theSquare, thePat);
    FrameRect(theSquare);
  end; (of DrawSquare)

  (=====)
  (Takes a point and returns vertical and horizontal values)
  procedure Unconvert (thePoint: Point; range: integer; var
    value, column: integer);
  var
    Lvalue, Lrange: longint;
  begin
    if thePoint.h < 5 then (out of range)
      column := -1
    else (get column)
      column := (thePoint.h - 5) div 10;
    if column > 22 then (out of range)
      column := -1;
    if (thePoint.v < 5) or (thePoint.v > 260) then
      value := -1 (out of range)
    else
      begin (get value)
        Lvalue := thePoint.v - 5;
        Lrange := range;
        Lvalue := Lvalue * Lrange;
        value := Lvalue div 255;
      end;
    end; (of unconvert)

  (=====)

```

```

(returns vertical position)
function Convert (value, range: integer): integer;
var
    Lvalue, Lrange: longint;
begin
    Lvalue := value;
    Lrange := range;
    Convert := ((Lvalue * 245) div Lrange) + 5;
end; (of Convert)

(=====)
(Update our window, someone uncovered a part of us)
procedure UpDate_Untitled;
var
    SavePort: WindowPtr; (Place to save the last port)
    theValue, theTop, i, rangeStop, RangeStart: integer;
begin
    if (MyWindow <> nil) and (MyWindow = whichWindow) then
        begin
            GetPort(SavePort); (Save the current port)
            SetPort(MyWindow); (Set the port to my window)
            TextFont(systemFont); (Set the font to draw in)

            ( Draw DrawGraph Stuff )
            SetRect(TempRect, 245, 20, 355, 105);
            FrameRect(TempRect); (Frame this rectangle area)
            SetRect(tempRect, 265, 5, 345, 20);
            sTemp := 'Draw Graph';
            TextBox(Pointer(ord(@sTemp) + 1), length(sTemp),
tempRect, teJustLeft);
            SetRect(tempRect, 248, 28, 258, 38);
            FillRect(tempRect, black);
            FrameRect(tempRect);
            SetRect(tempRect, 248, 48, 258, 58);
            FillRect(tempRect, dkgray);
            FrameRect(tempRect);
            SetRect(tempRect, 248, 68, 258, 78);
            FillRect(tempRect, gray);
            FrameRect(tempRect);
            SetRect(tempRect, 248, 88, 258, 98);
            FillRect(tempRect, ltgray);
            FrameRect(tempRect);
            FrameRect(tempRect);
            TextBox(Pointer(ord(@FreqText) + 1),
length(FreqText), FreqRect, teJustLeft);
            TextBox(Pointer(ord(@AmpText) + 1), length(AmpText),
AmpRect, teJustLeft);
            TextBox(Pointer(ord(@DurText) + 1), length(DurText),
DurRect, teJustLeft);
            TextBox(Pointer(ord(@TimbreText) + 1),
length(TimbreText), TimbreRect, teJustLeft);

            ( Draw a rectangle, ViewGraphRect )
            SetRect(TempRect, 390, 190, 490,
275); (left,top,right,bottom)
            FrameRect(TempRect); (Frame this rectangle area)
            SetRect(tempRect, 400, 175, 475, 190);
            sTemp := 'View Graph';
            TextBox(Pointer(ord(@sTemp) + 1), length(sTemp),
tempRect, teJustLeft);

            ( Draw a rectangle, NotePalette )
            PenPat(white);
            PaintRect(NotePalette);
            PenNormal;
            FrameRect(NotePalette); (Frame this rectangle area)
            RangeStart := GetCtlValue(C_NoteScrollbar);
            RangeStop := RangeStart + 22;
            if RangeStop > 100 then
                RangeStop := 100;
            for i := RangeStart to RangeStop do
                begin
                    if GetCtlValue(C_Timbre) = 1 then
                        begin

```

```

                            theValue := MyDoc^.timbre[i];
                            theTop := Convert(theValue, 254);
                            DrawSquare(theTop, (i - RangeStart) * 10 + 5, ltgray);
                        end;
                    if GetCtlValue(C_Duration) = 1 then
                        begin
                            theValue := MyDoc^.dur[i];
                            theTop := Convert(theValue, 250);
                            DrawSquare(theTop, (i - RangeStart) * 10 + 5, gray);
                        end;
                    if GetCtlValue(C_Amplitude) = 1 then
                        begin
                            theValue := MyDoc^.amp[i];
                            theTop := Convert(theValue, 255);
                            DrawSquare(theTop, (i - RangeStart) * 10 + 5, dkgray);
                        end;
                    if GetCtlValue(C_Frequency) = 1 then
                        begin
                            theValue := MyDoc^.freq[i];
                            theTop := Convert(theValue, 128);
                            DrawSquare(theTop, (i - RangeStart) * 10 + 5, black);
                        end;
                    end;

                    (Music Selection Stuff)
                    SetRect(tempRect, 255, 175, 365, 190);
                    sTemp := 'Music Selection';
                    TextBox(Pointer(ord(@sTemp) + 1), length(sTemp),
tempRect, teJustLeft);
                    SetRect(TempRect, 245, 190, 375, 275);
                    FrameRect(TempRect); (Frame this rectangle area)
                    TextBox(Pointer(ord(@EndText) + 1), length(EndText),
EndRect, teJustLeft);
                    TextBox(Pointer(ord(@StartText) + 1),
length(StartText), StartRect, teJustLeft);
                    SetRect(tempRect, 250, 235, 280, 250);
                    sTemp := 'End: ';
                    TextBox(Pointer(ord(@sTemp) + 1), length(sTemp),
tempRect, teJustLeft);
                    SetRect(tempRect, 250, 195, 290, 210);
                    sTemp := 'Start: ';
                    TextBox(Pointer(ord(@sTemp) + 1), length(sTemp),
tempRect, teJustLeft);

                    (Draw NoteIndex)
                    SetRect(tempRect, 370, 5, 410, 20);
                    sTemp := 'Note: ';
                    TextBox(Pointer(ord(@sTemp) + 1), length(sTemp),
tempRect, teJustLeft);
                    TextBox(Pointer(ord(@NoteText) + 1),
length(NoteText), NoteRect, teJustLeft);
                    TextFont(applFont); (Set the default application font)

                    DrawControls(MyWindow); (Draw all the controls)
                    SetPort(SavePort); (Restore the old port)
                end; (End for if (MyWindow <> nil))
            end; (End of UpDate_Untitled)

(=====)
(Open our window and draw everything)
procedure Open_Untitled;
begin
    if (MyWindow = nil) then
        begin
            MyWindow := GetNewWindow(1, nil, Pointer(-1));
            SetPort(MyWindow);
            ( Make a button, Play Sound )
            C_Play_Sound := GetNewControl(CB_Play_Sound, MyWindow);
            ( Make a checkboxes )
            C_Timbre := GetNewControl(CB_Timbre, MyWindow);
            C_Duration := GetNewControl(CB_Duration, MyWindow);
            C_Amplitude := GetNewControl(CB_Amplitude, MyWindow);
            C_Frequency := GetNewControl(CB_Frequency, MyWindow);
            ( Make a radio buttons )

```



```

R1Control[4] := GetNewControl(RB_Timbre, MyWindow);
R1Control[3] := GetNewControl(RB_Duration, MyWindow);
R1Control[2] := GetNewControl(RB_Amplitude, MyWindow);
R1Control[1] := GetNewControl(RB_Frequency, MyWindow);
( Make a scroll bars )
C_DurationScrollbar :=
GetNewControl(I_DurationScrollbar, MyWindow);
SetCtlValue(C_DurationScrollbar,
MyDoc^.dur[NoteIndex]);
C_AmplitudeScrollbar :=
GetNewControl(I_AmplitudeScrollbar, MyWindow);
SetCtlValue(C_AmplitudeScrollbar,
MyDoc^.amp[NoteIndex]);
C_FrequencyScrollbar :=
GetNewControl(I_FrequencyScrollbar, MyWindow);
SetCtlValue(C_FrequencyScrollbar,
MyDoc^.freq[NoteIndex]);
C_TimbreScrollbar := GetNewControl(I_TimbreScrollbar,
MyWindow);
SetCtlValue(C_TimbreScrollbar,
MyDoc^.timbre[NoteIndex]);
C_EndScrollbar := GetNewControl(I_EndScrollbar, MyWindow);
C_StartScrollbar := GetNewControl(I_StartScrollbar, MyWindow);
SetCtlValue(C_EndScrollbar, MyDoc^.EndValue);
SetCtlValue(C_StartScrollbar, MyDoc^.StartValue);
C_NoteScrollbar := GetNewControl(I_NoteScrollbar,
MyWindow);
ShowWindow(MyWindow);
SelectWindow(MyWindow);
end
else
SelectWindow(MyWindow);
end; (End of Open_Untitled)

(=====)
(Handle action to our window, like controls)
procedure Do_Untitled;
var
RefCon: longint;
code: integer;
theValue: integer;
whichWindow: WindowPtr;
myPt: Point;
theControl: ControlHandle;
MyErr: OSErr;
tempRect: rect;
newValue, NewPosition: integer;

procedure Do_A_Button;
begin
HiliteControl(theControl, 10);
RefCon := GetCRefCon(theControl);
case RefCon of
(Select correct button)
B_Play_Sound: (Play Sound, button)
begin (start for this button)
CreateSndResource(GetCtlValue(C_StartScrollbar),
GetCtlValue(C_EndScrollbar));
MyErr := SndPlay(nil, MyHandle, false);
DisposHandle(MyHandle);
DisposHandle(Handle(MySoundHandle));
MyHandle := nil;
MySoundHandle := nil;
end; (end for this button)
otherwise
begin (start)
end; (end of otherwise)
end; (end of case)
HiliteControl(theControl, 0); (Lighten the button)
end; (Handle a button being pressed)

procedure Do_A_Checkbox;
var
Index: integer; (Index used for radios)
procedure ClearIRadioGroup;

```

```

var
Index: integer; (Index used for radios)
begin (Start of the clear routine)
for Index := 1 to 4 do (Step thru all radios)
SetCtlValue(R1Control[Index], 0);
end; (End of the clear routine)
begin (Handle a checkbox being pressed)
RefCon := GetCRefCon(theControl);
theValue := GetCtlValue(theControl);
theValue := (theValue + 1) mod 2;
InvalRect(NotePallete);
case RefCon of
CB_Timbre, CB_Duration, CB_Amplitude, CB_Frequency:
begin
SetCtlValue(theControl, theValue);
end; (end for this checkbox)
RB_Timbre: (Timbre , radio button)
begin (start for this radio button)
DrawTool := 4;
ClearIRadioGroup;
SetCtlValue(theControl, 1); (Select this Radio)
SetCtlValue(C_Timbre, 1);
end;
RB_Duration: (Duration , radio button)
begin (start for this radio button)
DrawTool := 3;
ClearIRadioGroup; (Clear Radio values)
SetCtlValue(theControl, 1); (Select this Radio)
SetCtlValue(C_Duration, 1);
end;
RB_Amplitude: (Amplitude , radio button)
begin (start for this radio button)
DrawTool := 2;
ClearIRadioGroup;
SetCtlValue(theControl, 1); (Select this Radio)
SetCtlValue(C_Amplitude, 1);
end;
RB_Frequency: (Frequency , radio button)
begin (start for this radio button)
DrawTool := 1;
ClearIRadioGroup;
SetCtlValue(theControl, 1); (Select this Radio)
SetCtlValue(C_Frequency, 1);
end;
otherwise
begin
end;
end; (end of case)
end; (Handle a checkbox being pressed)

procedure Do_A_ScrollBar (code: integer);
procedure HandleWScrollbar (code, Start, Stop,
Increment, LIncrement: integer; theControl: ControlHandle);
var
theValue: integer; (Value of the scrollbar)
MaxTick: longint; (Timer for repeat scrolling)
FirstTime: boolean; (Flag to start scrolling)
begin
FirstTime := TRUE;
while (StillDown or FirstTime) do
begin (Timer used for repeat scrolling)
FirstTime := FALSE;
HiliteControl(theControl, code);
theValue := GetCtlValue(theControl);
if (code = inUpButton) then
begin
theValue := theValue - Increment;
if (theValue < Start) then
theValue := Start;
end;
if (code = inDownButton) then
begin
theValue := theValue + Increment;
if (theValue > Stop) then

```

```

        theValue := Stop; (Bump at the stop value)
    end;
    if (code = inPageUp) then
    begin
        theValue := theValue - LIncrement;
        if (theValue < Start) then
            theValue := Start;
        end;
    if (code = inPageDown) then
    begin
        theValue := theValue + LIncrement;
        if (theValue > Stop) then
            theValue := Stop; (Bump at the Stop value)
        end;
    if (code = inThumb) then
    begin
        code := TrackControl(theControl, myPt,
nil); (Let the OS drag it around)
        theValue := GetCtlValue(theControl);
    end;
    SetCtlValue(theControl, theValue); (Set new state)
    MaxTick := TickCount + 9;
    repeat (Start of delay routine)
    until not (Button) or (TickCount >
MaxTick); (Exit when time up or mouse button up)
    HiliteControl(theControl, 0); (Lighten the arrow)
    end; (End for StillDown)
end; (End of handle scrollbar)

begin (Handle a ScrollBar being pressed)
RefCon := GetRefCon(theControl); (get control refcon)
TempRect := NotePalette;
TempRect.left := 6;
TempRect.right := TempRect.left + 8;
case RefCon of (Select correct scrollbar)
    I_DurationScrollbar: (DurationScrollbar, scroll bar)
    begin
        dirty := true;
        HandleWScrollbar(code, 0, 250, 1, 10, theControl);
        theValue := GetCtlValue(theControl);
        MyDoc^.dur[NoteIndex] := theValue;
        NumToString(theValue, DurText);
        InvalRect(DurRect);
    end;
    I_AmplitudeScrollbar: (AmplitudeScrollbar, scroll bar)
    begin
        dirty := true;
        HandleWScrollbar(code, 0, 255, 1, 10, theControl);
        theValue := GetCtlValue(theControl);
        MyDoc^.amp[NoteIndex] := theValue;
        NumToString(theValue, AmpText);
        InvalRect(AmpRect);
    end;
    I_FrequencyScrollbar: (FrequencyScrollbar, scroll bar)
    begin
        dirty := true;
        HandleWScrollbar(code, 0, 128, 1, 10, theControl);
        theValue := GetCtlValue(theControl);
        MyDoc^.freq[NoteIndex] := theValue;
        if theValue < 128 then
            NumToString(theValue, FreqText)
        else
            FreqText := 'Rest';
            InvalRect(FreqRect);
        end;
    end;
    I_TimbreScrollbar: (TimbreScrollbar, scroll bar)
    begin
        dirty := true;
        HandleWScrollbar(code, 0, 254, 1, 10, theControl);
        theValue := GetCtlValue(theControl);
        MyDoc^.timbre[NoteIndex] := theValue;
        NumToString(theValue, TimbreText);
        InvalRect(TimbreRect);
    end;
end;

```

```

I_EndScrollbar: (EndScrollbar, scroll bar)
begin
    dirty := true;
    HandleWScrollbar(code, 1, 100, 1, 10, theControl);
    GetCtlValue(C_StartScrollbar, 100, 1, 10, theControl);
    theValue := GetCtlValue(theControl);
    MyDoc^.EndValue := theValue;
    NumToString(theValue, EndText);
    TempRect := EndRect;
end;
I_StartScrollbar: (StartScrollbar, scroll bar)
begin (start for this scrollbar)
    dirty := true;
    HandleWScrollbar(code, 1, 100, 1, 10, theControl);
    theValue := GetCtlValue(theControl);
    MyDoc^.StartValue := theValue;
    NumToString(theValue, StartText);
    InvalRect(StartRect);
    SetCtlMin(C_EndScrollbar, theValue);
    theValue := GetCtlValue(C_EndScrollbar);
    MyDoc^.EndValue := theValue;
    NumToString(theValue, EndText);
    TempRect := EndRect;
end;
I_NoteScrollbar: (NoteScrollbar, scroll bar)
begin
    HandleWScrollbar(code, 1, 100, 1, 10, theControl);
    theValue := GetCtlValue(theControl);
    NoteIndex := theValue;
    NumToString(theValue, NoteText);
    InvalRect(NoteRect);
    SetCtlValue(C_DurationScrollbar, MyDoc^.dur[NoteIndex]);
    SetCtlValue(C_FrequencyScrollbar, MyDoc^.freq[NoteIndex]);
    SetCtlValue(C_AmplitudeScrollbar, MyDoc^.amp[NoteIndex]);
    SetCtlValue(C_TimbreScrollbar, MyDoc^.timbre[NoteIndex]);
    NumToString(MyDoc^.timbre[NoteIndex], TimbreText);
    NumToString(MyDoc^.dur[NoteIndex], durText);
    NumToString(MyDoc^.amp[NoteIndex], ampText);
    if MyDoc^.freq[NoteIndex] < 128 then
        NumToString(MyDoc^.freq[NoteIndex], FreqText)
    else
        FreqText := 'Rest';
        InvalRect(durRect);
        InvalRect(freqRect);
        InvalRect(ampRect);
        InvalRect(timbreRect);
        TempRect := NotePalette;
        InsetRect(TempRect, 1, 1);
    end; (end for this scrollbar)
otherwise
begin
end;
end; (end of case)
InvalRect(TempRect);
end; (Handle a ScrollBar being pressed)

begin (Start of Window handler)
    if (MyWindow <> nil) then
    begin
        code := FindWindow(myEvent.where, whichWindow);
        if (myEvent.what =MouseDown) and (MyWindow =
whichWindow) then
        begin
            myPt := myEvent.where; (Get mouse position)
            GlobalToLocal(myPt);
        end;
        if (MyWindow = whichWindow) and (code = inContent)
        then
        begin
            code := FindControl(myPt, whichWindow, theCon-
trol);
            if (code = inUpButton) or (code = inDownButton)
            or (code = inThumb) or (code = inPageDown) or (code =
inPageUp) then

```

```

        Do_A_ScrollBar(code); (Do scrollbars)
        if (code <> 0) then (Check type of control)
            code := TrackControl(theControl, myPt,
nil); (Track the control)
        if code = inButton then
            Do_A_Button; (Do buttons)
        if code = inCheckBox then
            Do_A_Checkbox; (Do checkboxes)
        if PtInRect(myPt, NotePalette) then
            repeat
                dirty := true;
                case DrawTool of
                    1:
                        UnConvert(myPt, 128, newValue, NewPosition);
                    2:
                        UnConvert(myPt, 255, newValue, NewPosition);
                    3:
                        UnConvert(myPt, 250, newValue, NewPosition);
                    4:
                        UnConvert(myPt, 254, newValue, NewPosition);
                end;
                if (newValue <> -1) and (newPosition <> -1)
then
                    begin (still in NotePalette?)
                        case DrawTool of
                            1:
                                begin
                                    MyDoc^.freq[GetCtlValue(C_NoteScrollbar) + NewPosition] :=
newValue;
                                    if NewPosition = 0 then
                                        begin
                                            SetCtlValue(C_FrequencyScrollbar, newValue);
                                            if newValue <> 128 then
                                                NumToString(newValue, FreqText);
                                            else
                                                FreqText := 'Rest';
                                                InvalRect(FreqRect);
                                            end;
                                        end;
                                    2:
                                        begin
                                            MyDoc^.amp[GetCtlValue(C_NoteScrollbar)
+ NewPosition] := newValue;
                                            if NewPosition = 0 then
                                                begin
                                                    SetCtlValue(C_AmplitudeScrollbar, newValue);
                                                    NumToString(newValue, AmpText);
                                                    InvalRect(AmpRect);
                                                end;
                                            end;
                                    3:
                                        begin
                                            MyDoc^.dur[GetCtlValue(C_NoteScrollbar)
+ NewPosition] := newValue;
                                            if NewPosition = 0 then
                                                begin
                                                    SetCtlValue(C_DurationScrollbar,
newValue);
                                                    NumToString(newValue, DurText);
                                                    InvalRect(DurRect);
                                                end;
                                            end;
                                    4:
                                        begin
                                            MyDoc^.timbre[GetCtlValue(C_NoteScrollbar) + NewPosition] :=
newValue;
                                            if NewPosition = 0 then
                                                begin
                                                    SetCtlValue(C_TimbreScrollbar,
newValue);
                                                    NumToString(newValue, TimbreText);
                                                    InvalRect(TimbreRect);
                                                end;
                                            end;
                                        end;
                                end;
                            end;
                        end;
                    end;
                end;
            end;
        end;
    end;
end;

```

```

        end; (of Case)
        SetRect(TempRect, (NewPosition * 10 +
6), 6, (NewPosition * 10 + 5) + 9, 259);
        InvalRect(TempRect);
        end; (end of still in NotePalette)
        BeginUpdate(MyWindow);
        Update_Untitled(MyWindow);
        EndUpdate(MyWindow);
        GetMouse(myPt);
        until not (StillDown);
        end; (End for if (MyWindow=whichWindow))
        end; (End for if (MyWindow<>nil))
        end; (End of procedure)
    end; (End of unit)
end.

```

Listing: HandleTheMenus.pas

```

unit HandleTheMenus;
interface
uses
    PrintTraps, Message, save_changes, About, Untitled, Sound,
    MyGlobals, MyFileStuff, MyPrintStuff;
procedure AdjustMenus;
procedure Handle_My_Menu (var doneFlag: boolean; theMenu,
theItem: integer); (Handle menu selection)
implementation
procedure AdjustMenus;
begin
    if (FrontWindow <> MyWindow) then
        begin (Something up there)
            DisableItem(M_Extend, 0);
            EnableItem(M_Edit, 0);
            DisableItem(M_File, MI_Open);
            DisableItem(M_File, MI_New);
            DisableItem(M_File, MI_Close);
            DisableItem(M_File, MI_Save);
            DisableItem(M_File, MI_Save_As);
            DisableItem(M_File, MI_Page_Setup);
            DisableItem(M_File, MI_Print);
        end
    else if MyWindow <> nil then
        begin (My Window up there)
            EnableItem(M_Extend, 0);
            DisableItem(M_Edit, 0);
            DisableItem(M_File, MI_Open);
            DisableItem(M_File, MI_New);
            EnableItem(M_File, MI_Close);
            EnableItem(M_File, MI_Save);
            EnableItem(M_File, MI_Save_As);
            EnableItem(M_File, MI_Page_Setup);
            EnableItem(M_File, MI_Print);
        end
    else
        begin (nothing up there)
            VolRefNum := 0; (no need to save any changes)
            DisableItem(M_Extend, 0);
            DisableItem(M_Edit, 0);
            EnableItem(M_File, MI_Open);
            EnableItem(M_File, MI_New);
            DisableItem(M_File, MI_Close);
            DisableItem(M_File, MI_Save);
            DisableItem(M_File, MI_Save_As);
            DisableItem(M_File, MI_Page_Setup);
            DisableItem(M_File, MI_Print);
        end;
    end;
end;
procedure Handle_My_Menu;
var
    DNA: integer;
    BoolHolder: boolean;
    DAName, title: Str255;
    SavePort: GrafPtr;
    i, theValue: integer;
begin
    (Start of procedure)
    case theMenu of (Do selected menu list)

```

```

L_Apple:
begin
  case theItem of
    MI_About_Zoundz:
      begin
        D_About;
      end;
    otherwise(Handle the DAs)
      begin
        GetPort(SavePort);
        GetItem(AppleMenu, theItem, DAName);
        DNA := OpenDeskAcc(DAName);
        SetPort(SavePort);
      end;
  end; (End of item case)
end; (End for Apple Menu list)

L_File:
begin
  case theItem of
    MI_New:
      begin
        Init_Untitled;
        Open_Untitled;
        dirty := false;
      end;
    MI_Open:
      begin
        doOpen;
        if VolRefNum <> 0 then
          begin
            Open_Untitled;
            SetWTitle(MyWindow, FileName);
            dirty := false;
          end;
        end;
      end;
    MI_Close:
      begin
        if dirty then
          begin (need saving)
            GetWTitle(MyWindow, title);
            ParamText(title, '', '', '');
            theValue := D_save_changes;
            if (theValue = I_Yes) then
              begin (wants to save changes)
                if VolRefNum = 0 then
                  doSaveAs
                else
                  doSave;
                if VolRefNum <> 0 then
                  Close_Untitled(MyWindow);
                end
              end
            else if theValue = I_No then
              Close_Untitled(MyWindow);
            end (of needing saved)
          else (no need to save)
            Close_Untitled(MyWindow);
          end;
        end;
      end;
    MI_Save:
      begin
        if VolRefNum = 0 then
          doSaveAs
        else
          doSave;
        if VolRefNum <> 0 then
          dirty := false;
        end;
      end;
    MI_Save_As:
      begin
        doSaveAs;
        if VolRefNum <> 0 then
          dirty := false;
        end;
      end;
    MI_Page_Setup:
      begin

```

```

        doSetup;
      end;
    MI_Print:
      begin
        doPrint;
      end;
    MI_Quit:
      begin
        if (MyWindow <> nil) and dirty then
          begin (need saving)
            GetWTitle(MyWindow, title);
            ParamText(title, '', '', '');
            theValue := D_save_changes;
            case theValue of
              I_Yes:
                begin
                  if VolRefNum = 0 then
                    doSaveAs
                  else
                    doSave;
                  if VolRefNum <> 0 then
                    doneFlag := TRUE;
                  end; (of Yes)
                end
              I_No:
                doneFlag := True;
              otherwise
                ;
            end; (of case)
          end (of needing saved)
        else (no need to save)
          doneFlag := TRUE;
        end;
      end;
    otherwise
      ;
    end; (of item list)
  end; (of File Menu List )

L_Edit:
begin
  BoolHolder := SystemEdit(theItem - 1); (DA Editing)
end;

L_Extend:
begin
  case theItem of(Handle extending values)
    MI_Frequency:
      begin
        theValue := MyDoc^.freq[NoteIndex];
        for i := MyDoc^.StartValue to
MyDoc^.EndValue do
          MyDoc^.freq[i] := theValue;
        end; (of Frequency)
      end;
    MI_Amplitude:
      begin
        theValue := MyDoc^.amp[NoteIndex];
        for i := MyDoc^.StartValue to
MyDoc^.EndValue do
          MyDoc^.amp[i] := theValue;
        end; (of Amplitude)
      end;
    MI_Duration:
      begin
        theValue := MyDoc^.dur[NoteIndex];
        for i := MyDoc^.StartValue to
MyDoc^.EndValue do
          MyDoc^.dur[i] := theValue;
        end; (of duration)
      end;
    MI_Timbre:
      begin
        theValue := MyDoc^.timbre[NoteIndex];
        for i := MyDoc^.StartValue to
MyDoc^.EndValue do
          MyDoc^.timbre[i] := theValue;
        end; (of timbre)
      end;
    otherwise
      ;
  end; ( of items in Extend Menu)
end;

```

```

        InvalRect(NotePalette);
    end; (of Extend Menu List)
    otherwise
    ;
end; (End for the Menus)

HiliteMenu(0); (Turn menu selection off)
end; (End of procedure Handle_My_Menu)
end. (End of unit)

Listing: Zoundz.pas
program Zoundz;
(Program name: Zoundz.Pas )
(Function: Allows creation of 'snd' resource ID=9000. )
(History: 4/17/89 Original by Prototyper. )
(Modified: 5/1/89 by Kirk Chase)
uses
    PrintTraps, Message, save_changes, About, Untitled,
    InitTheMenus, HandleTheMenus, Sound, MyGlobals, MyFileStuff,
    MyPrintStuff;
const
    WNETrapNum = $60;
    UnimplTrapNum = $9F;
    MultiEvt = 15;
    bit0 = 31;
    GrayRgnLowMemGlobal = $9EE;
var
    (Main variables)
    myEvent: EventRecord;
    ResumePeek: WindowPeek;
    theWorld: SysEnvRec;
    doneFlag, DoIt, WNE, sysResult: boolean;
    code, theValue: integer;
    whichWindow: WindowPtr;
    dragRect: Rect;
    mResult: longint;
    theMenu, theItem: integer;
    chCode: integer;
    ch: char;
    myPt: Point;
    title: str255;
    nDocs, message, index, sleep: integer;
    theFile: AppFile;
    theErr: OSErr;

procedure InitMac; (initializes Macintosh)
begin
    MoreMasters; (This reserves space for more handles)
    InitGraf(@thePort); (Quickdraw Init)
    InitFonts; (Font manager init)
    InitWindows; (Window manager init)
    InitMenus; (Menu manager init)
    TEInit; (Text edit init)
    InitDialogs(nil); (Dialog manager)
    FlushEvents(everyEvent, 0); (Clear out all events)
    InitCursor; (Make an arrow cursor)
end;

procedure InitApp; (initialize application)
var
    tempHandle: handle;
begin
    doneFlag := FALSE; (Do not exit program yet)
    Init_My_Menus; (Initialize menu bar)
    MyDoc := DocPtr(NewPtr(sizeof(DocRec))); (get doc)
    ThePrintRec := nil; (print initialization)
    PrOpen;
    tempHandle := NewHandle(sizeof(TPrint));
    ThePrintRec := THPrint(tempHandle);
    PrintDefault(ThePrintRec);
    PageRect := ThePrintRec^.prInfo.rpage;
    PrClose;
    dragRect := screenbits.bounds; (Get drag area)
    SetRect(dragRect, dragRect.Left + 10, dragRect.Top + 25,
    dragRect.Right - 10, dragRect.Bottom - 10);

```

```

    SetRect(NotePalette, 5, 5, 235, 260); (set rectangles)
    SetRect(EndRect, 345, 235, 370, 250);
    SetRect(StartRect, 345, 195, 370, 210);
    SetRect(TimbreRect, 465, 85, 490, 100);
    SetRect(DurRect, 465, 65, 490, 80);
    SetRect(AmpRect, 465, 45, 490, 60);
    SetRect(FreqRect, 465, 25, 500, 40);
    SetRect(NoteRect, 420, 5, 445, 20);
    Dirty := false;
    theWatch := GetCursor(watchCursor);
    HLock(Handle(theWatch));
    theSquare := GetCursor(crossCursor);
    HLock(Handle(theSquare));
    theErr := SysEnvirons(1, theWorld);
    if (theWorld.machineType >= 0) and
    (NGetTrapAddress(WNETrapNum, ToolTrap) =
    NGetTrapAddress(UnimplTrapNum, ToolTrap)) then
        WNE := false
    else
        WNE := true;
        sleep := 10;
    end;

procedure AdjustCursor; (set cursor)
begin
    if (FrontWindow = MyWindow) and (MyWindow <> nil) then
        begin (our window in front)
            GetMouse(myPt);
            if PtInRect(myPt, NotePalette) then
                SetCursor(theSquare) (over note palette)
            else
                SetCursor(arrow); (over other stuff)
            end
        else
            SetCursor(arrow);
        end;
end;

begin (Start of main body)
    InitMac;
    InitApp;
    (finder startup)
    CountAppFiles(message, nDocs);
    if nDocs = 0 then
        begin (no files to open)
            Init_Untitled; (Initialize the window routines)
            Open_Untitled; (Open window routines at program start)
        end
    else
        begin (files to print or open)
            if message = appPrint then
                begin (print docs)
                    for index := 1 to nDocs do
                        begin (Loop through docs)
                            GetAppFiles(index, theFile);
                            if theFile.fType = 'ZZDC' then
                                begin (my file)
                                    VolRefNum := theFile.vRefNum;
                                    FileName := theFile.fName;
                                    OpenFile;
                                    if VolRefNum <> 0 then
                                        begin
                                            Open_Untitled;
                                            SetWTitle(MyWindow, FileName);
                                            doPrint;
                                            Close_Untitled(MyWindow);
                                        end;
                                    end;
                                end;
                                ClrAppFiles(index);
                            end; (Loop through docs)
                        doneFlag := true; (quit when done)
                    end (print docs)
                else
                    begin (open first file, can't open multiple)
                        GetAppFiles(1, theFile);

```

```

if theFile.fType = 'ZZDC' then
begin
  VolRefNum := theFile.vRefNum;
  FileName := theFile.fName;
  OpenFile;
  if VolRefNum = 0 then
  begin
    Init_Untitled; (Init window routines)
    FileName := 'Untitled';
  end;
  Open_Untitled; (Open window routines)
  SetWTitle(MyWindow, FileName);
  for index := 1 to nDocs do
    ClrAppFiles(index);
  end;
end;
end;

repeat (Start of main event loop)
  AdjustMenus;
  AdjustCursor;
  if WNE then
    DoIt := WaitNextEvent(everyEvent, myEvent, sleep, nil)
  else
  begin
    SystemTask;
    DoIt := GetNextEvent(everyEvent, myEvent);
  end;
  if DoIt then (If event then...)
  begin
    (Start handling the event)
    code := FindWindow(myEvent.where, whichWindow);
    case myEvent.what of (Decide type of event)
     MouseDown: (Mouse button pressed)
      begin
        if (code = inMenuBar) then
        begin
          mResult := MenuSelect(myEvent.Where);
          theMenu := HiWord(mResult);
          theItem := LoWord(mResult);
          Handle_My_Menu(doneFlag, theMenu, theItem);
        end; (End of inMenuBar)
        if (code = InDrag) then
        begin
          DragWindow(whichWindow, myEvent.where, dragRect);
        end;
        if (code = inGoAway) then
        begin
          if TrackGoAway(whichWindow, myEvent.where) then
          begin
            if dirty then
            begin (need saving)
              GetWTitle(MyWindow, title);
              ParamText(title, '', '', '');
              theValue := D_SaveChanges;
              if (theValue = I_Yes) then
              begin (wants to save changes)
                if VolRefNum = 0 then
                  doSaveAs
                else
                  doSave;
                if VolRefNum <> 0 then
                  Close_Untitled(MyWindow);
              end
            else if theValue = I_No then
              Close_Untitled(MyWindow);
            end (of needing saved)
          else (no need to save)
            Close_Untitled(MyWindow);
          end;
        end; (End of InGoAway)
        if (code = inContent) then
        begin
          if (whichWindow <> FrontWindow) then
            SelectWindow(whichWindow)

```

```

      else
      begin
        SetPort(whichWindow);
        Do_Untitled(myEvent);
      end;
    end; (End of inContent)
    if (code = inSysWindow) then
      SystemClick(myEvent, whichWindow);
    end; (End of MouseDown)
    KeyDown, AutoKey: (Handle key inputs)
    begin
      with myevent do
      begin
        chCode := BitAnd(message, CharCodeMask);
        ch := CHR(chCode);
        if (Odd(modifiers div CmdKey)) then
        begin
          mResult := MenuKey(ch);
          theMenu := HiWord(mResult);
          theItem := LoWord(mResult);
          if (theMenu <> 0) then
            Handle_My_Menu(doneFlag, theMenu, theItem);
          end;
        end;
      end; (End for KeyDown, AutoKey)
      UpDateEvt: (Update event for a window)
      begin
        whichWindow := WindowPtr(myEvent.message);
        BeginUpdate(whichWindow);
        Update_Untitled(whichWindow);
        EndUpdate(whichWindow);
      end; (End of UpDateEvt)
      DiskEvt: (Disk inserted event)
      begin
        if (HiWord(myevent.message) <> noErr) then
        begin (due to unformatted diskette inserted)
          myEvent.where.h :=
            ((screenbits.bounds.Right - screenbits.bounds.Left) div 2) -
            (304 div 2);
          myEvent.where.v :=
            ((screenbits.bounds.Bottom - screenbits.bounds.Top) div 3) -
            (104 div 2);
          InitCursor;
          chCode := DIBadMount(myEvent.where,
            myevent.message); (Let the OS handle the diskette)
          end;
        end; (End of DiskEvt)
        ActivateEvt: (Window activated event)
        begin
          whichWindow := WindowPtr(myevent.message);
          if odd(myEvent.modifiers) then
          begin (Handle the activate)
            SelectWindow(whichWindow);
          end;
        end; (End of ActivateEvt)
        MultiEvt:
        begin
          if Odd(myEvent.message) then
          begin (resume event)
            if FrontWindow = MyWindow then
            begin
              SetPort(MyWindow);
              InvalRect(MyWindow^.portRect);
            end
          else if FrontWindow <> nil then
          begin
            ResumePeek := WindowPeek(FrontWindow);
            if ResumePeek^.windowKind < 0 then
            begin
              myEvent.what := activateEvt;
              BitSet(@myEvent.modifiers, bit0);
              sysResult :=
                SystemEvent(myEvent);
            end;
          end;
        end;
      end;
    end;
  end;
end;

```

```

        end;
    end (of resume event)
else (suspend event)
begin
    if FrontWindow = MyWindow then
    begin
        SetPort(MyWindow);
        InvalRect(MyWindow^.portRect);
    end
    else if FrontWindow <> nil then
    begin
        ResumePeek := WindowPeek(FrontWindow);
        if ResumePeek^.windowKind < 0 then
        begin
            myEvent.what := activateEvt;
            BitClr(@myEvent.modifiers, bit0);
            sysResult :=
SystemEvent(myEvent);
        end;
    end;
end;
end; (of MultiEvt)
otherwise
;
end; (End of case)
end; (end of GetNextEvent)
until doneFlag; (End of the event loop)
end. (End of the program)

```

Listing: Zoundz.r

* RMaker resource file sources.
* File: Zoundz.R

Zoundz.RSRC
????????

include CursIcons
include AppStuff

Type DLOG

```

    ,3      ;;Resource ID
save changes      ;;Dialog title
50 120 188 365    ;;Top Left Bottom Right
Visible NoGoAway  ;;Visible GoAway
1            ;;ProcID, dialog def ID
3            ;;Refcon, reference value
3            ;;ID of item list

```

Type DITL

```

    ,3      ;;Resource ID
4          ;;Number of controls in list

Button Enabled    ;;Push button
70 30 90 100      ;;Top Left Bottom Right
Yes              ;;message

Button Enabled    ;;Push button
100 150 120 220   ;;Top Left Bottom Right
Cancel           ;;message

Button Enabled    ;;Push button
100 30 120 100    ;;Top Left Bottom Right
No              ;;message

StaticText        ;;Static text
10 30 55 220      ;;Top Left Bottom Right
Save changes to ^0?  ;;message

```

Type WIND

```

    ,1      ;;Resource ID
Untitled        ;;Window title

```

```

42 5 337 506      ;;Top Left Bottom Right
InVisible GoAway  ;;Visible GoAway
4                ;;ProcID, Window def ID
1                ;;Refcon, reference value

Type CNTL

    ,26          ;;Resource ID
Play Sound        ;;Title for a Button
120 245 160 490   ;;Top Left Bottom Right
Visible          ;;Initially visible
0                ;;ProcID (Control definition ID)
26               ;;RefCon (reference value)
0 1 0            ;;Min Max Value

    ,16          ;;Resource ID
Timbre            ;;Title for a Checkbox
255 395 270 485   ;;Top Left Bottom Right
Visible          ;;Initially visible
1                ;;ProcID (Control definition ID)
16               ;;RefCon (reference value)
0 1 1            ;;Min Max Value

    ,15          ;;Resource ID
Duration          ;;Title for a Checkbox
235 395 250 485   ;;Top Left Bottom Right
Visible          ;;Initially visible
1                ;;ProcID (Control definition ID)
15               ;;RefCon (reference value)
0 1 1            ;;Min Max Value

    ,14          ;;Resource ID
Amplitude         ;;Title for a Checkbox
215 395 230 485   ;;Top Left Bottom Right
Visible          ;;Initially visible
1                ;;ProcID (Control definition ID)
14               ;;RefCon (reference value)
0 1 1            ;;Min Max Value

    ,13          ;;Resource ID
Frequency         ;;Title for a Checkbox
195 395 210 485   ;;Top Left Bottom Right
Visible          ;;Initially visible
1                ;;ProcID (Control definition ID)
13               ;;RefCon (reference value)
0 1 1            ;;Min Max Value

    ,20          ;;Resource ID
Timbre            ;;Title for a RadioButton
85 260 100 350    ;;Top Left Bottom Right
Visible          ;;Initially visible
2                ;;ProcID (Control definition ID)
20               ;;RefCon (reference value)
0 1 0            ;;Min Max Value

    ,19          ;;Resource ID
Duration          ;;Title for a RadioButton
65 260 80 350     ;;Top Left Bottom Right
Visible          ;;Initially visible
2                ;;ProcID (Control definition ID)
19               ;;RefCon (reference value)
0 1 0            ;;Min Max Value

    ,18          ;;Resource ID
Amplitude         ;;Title for a RadioButton
45 260 60 350     ;;Top Left Bottom Right
Visible          ;;Initially visible
2                ;;ProcID (Control definition ID)
18               ;;RefCon (reference value)
0 1 0            ;;Min Max Value

    ,17          ;;Resource ID
Frequency         ;;Title for a RadioButton
25 260 40 350     ;;Top Left Bottom Right

```

```

Visible                ;;Initially visible
2                      ;;ProcID (Control definition ID)
17                     ;;RefCon (reference value)
0 1 1                  ;;Min Max Value

,41                    ;;Resource ID
DurationScrollbar      ;;Title for a Scrollbar
65 360 81 460         ;;Top Left Bottom Right
Visible                ;;Initially visible
16                     ;;ProcID (Control definition ID)
41                     ;;RefCon (reference value)
0 250 0                ;;Min Max Value

,40                    ;;Resource ID
AmplitudeScrollbar     ;;Title for a Scrollbar
45 360 61 460         ;;Top Left Bottom Right
Visible                ;;Initially visible
16                     ;;ProcID (Control definition ID)
40                     ;;RefCon (reference value)
0 255 0                ;;Min Max Value

,39                    ;;Resource ID
FrequencyScrollbar     ;;Title for a Scrollbar
25 360 41 460         ;;Top Left Bottom Right
Visible                ;;Initially visible
16                     ;;ProcID (Control definition ID)
39                     ;;RefCon (reference value)
0 128 0                ;;Min Max Value

,38                    ;;Resource ID
TimbreScrollbar        ;;Title for a Scrollbar
85 360 101 460        ;;Top Left Bottom Right
Visible                ;;Initially visible
16                     ;;ProcID (Control definition ID)
38                     ;;RefCon (reference value)
0 254 0                ;;Min Max Value

,31                    ;;Resource ID
EndScrollbar           ;;Title for a Scrollbar
255 250 271 370       ;;Top Left Bottom Right
Visible                ;;Initially visible
16                     ;;ProcID (Control definition ID)
31                     ;;RefCon (reference value)
1 100 1                ;;Min Max Value

,27                    ;;Resource ID
StartScrollbar         ;;Title for a Scrollbar
215 250 231 370       ;;Top Left Bottom Right
Visible                ;;Initially visible
16                     ;;ProcID (Control definition ID)
27                     ;;RefCon (reference value)
1 100 1                ;;Min Max Value

,12                    ;;Resource ID
NoteScrollbar          ;;Title for a Scrollbar
266 5 282 231         ;;Top Left Bottom Right
Visible                ;;Initially visible
16                     ;;ProcID (Control definition ID)
12                     ;;RefCon (reference value)
1 100 1                ;;Min Max Value

Type ALRT

,4                      ;;Resource ID
75 124 210 385        ;;Top Left Bottom Right
4                      ;;ID of item list
CCCC                  ;;stages of alert in hexadecimal

Type DITL

,4                      ;;Resource ID
2                      ;;Number of controls in list

StaticText             ;;Static text

```

```

10 55 80 245          ;;Top Left Bottom Right
~0\0D~1\0D~2\0D~3    ;;message

Button Enabled         ;;Push button
90 140 110 200        ;;Top Left Bottom Right
OK                     ;;message

Type DLOG

,2                      ;;Resource ID
About                  ;;Dialog title
112 136 210 373       ;;Top Left Bottom Right
Visible NoGoAway       ;;Visible GoAway
1                      ;;ProcID, dialog def ID
2                      ;;Refcon, reference value
2                      ;;ID of item list

Type DITL

,2                      ;;Resource ID
3                      ;;Number of controls in list

Button Enabled         ;;Push button
60 75 87 155          ;;Top Left Bottom Right
OK                     ;;message

StaticText             ;;Static text
10 35 25 205          ;;Top Left Bottom Right
Zoundz 1.0 By Kirk Chase ;;message

StaticText             ;;Static text
35 10 50 225          ;;Top Left Bottom Right
© 1989 Kirk Chase and MacTutor ;;message

Type MENU

,1001                  ;;Resource ID
\14                    ;;APPLE menu title
About Zoundz...        ;;item title
(-                      ;;

,1002                  ;;Resource ID
File                   ;;menu title
(New/N                  ;;item title
(Open.../0             ;;item title
(-                      ;;
Close                  ;;item title
Save/S                 ;;item title
Save As...             ;;item title
(-                      ;;
Page Setup...          ;;item title
Print...               ;;item title
(-                      ;;
Quit/Q                 ;;item title

,1003                  ;;Resource ID
Edit                   ;;menu title
Undo/Z                 ;;item title
(-                      ;;
Cut/X                  ;;item title
Copy/C                 ;;item title
Paste/V                ;;item title

,1004                  ;;Resource ID
Extend                 ;;menu title
Frequency              ;;item title
Amplitude              ;;item title
Duration               ;;item title
Timbre                 ;;item title

```



Programmer's Forum

Understanding Color Icons



Steve and Patricia Sheets
Herdon, VA
Volume 5, Number 10

Understanding, Creating and Using Color Icons Steve and Patricia Sheets

Traditional Quickdraw contains numerous data structures. Bit Images, BitMaps, Patterns, and Icons are all graphic concepts that a Macintosh programmer uses in order to manipulate Quickdraw. Most of these graphic concepts are layered in construction. For example, in order to understand how to create and manipulate Icons, a Mac programmer needs to know about Bitmaps.

The topic of this column is the Color Icon data structure. The same layered approach that is used in traditional Quickdraw is also used in Color Quickdraw. Thus in order to understand Color Icons, this article will explain about Pixel Images, Pixel Bitmaps, and Color Look-up tables. Once the Color Icon's structure is defined, this article will discuss the simple usage of Color Icons in Menus and Dialogs. Finally, creating Color Icons using ResEdit templates will be covered.

Bit Images and Bitmaps

In Quickdraw, the basis of all graphic data structures is the Bit Image. The Bit Image is a portion of memory that represents an arbitrary black and white display. Each bit of the image represents a pixel of the display. If a Bit is set, the pixel is black. If it is unset, the pixel is white. The pixels are laid out left to right, then top to bottom. The Pixel in the upper left corner is defined in Bit 1, the Pixel to the right of it is defined in Bit 2 and so on. Notice that a Bit Image has no built-in definition. There is no explanation of the horizontal or vertical dimensions of the image. Non-color Macintosh (Mac 128K, Mac 512K, Mac Plus, Mac SE) screens are represented as Bit Images.

The Quickdraw Bitmap is the data structure that represents an exact black and white display. A Bitmap contains a pointer to the Bit Image, the row width of that Bit Image, and the coordinates of that Image. The coordinates of the Bit Image is defines as a boundary rectangle. The upper left coordinate of the rectangle is the first pixel of the Bitmap. In most cases, this is defined as 0,0. While it is easiest to work with Bitmaps laid out this way, the upper left coordinate could be any value. The row width defines the the number of pixels that are on one row of the Bit Image. This is given in Bytes of memory, not pixels. Also, the row width must be an even number of Bytes. Thus a Bitmap that had 32 pixels in a row would have a byte with at least 4 bytes (8*4). The row width could be larger, that would just mean some of the bits of the Image were not being used. For example, a Bitmap that had only 30 pixels in a row would still require a row width of 4 bytes. The 31st & 32nd bits would not be used.

```
Bitmap = RECORD
```

```
baseAddr: Ptr;  
rowBytes: Integer;  
bounds: Rect;  
END;
```

Pix Images and PixMaps

Color Quickdraw's equivalent of a Bit Image is the Pixel Image. The Pixel Image is a portion of memory that represents a display, possibly a color display. Each Pixel on the display is represented by one or more bits in the memory. The number of bits that is required for a single pixel is called the depth of the Image. Pixel Images are similar to Bit Images in that the size and row width of a Pixel Image is not defined. Also undefined is the depth of the image, the exact lay out of the bits in memory, and the conversion of the bits of a pixel to colors. All these values are defined in a Pixel Map.

The Pixel Map is Color Quickdraw's equivalent of a Bit Map. Like a Bitmap, it contains a pointer to the Image (Pixel in this case), the row width (in bytes) of the Image, and the dimensions of the rectangle. However the Pixel Map is more complicated than that.

```
PixelMap = RECORD  
baseAddr: Ptr;  
rowBytes: Integer;  
bounds: Rect;  
pmVersion: Integer;  
packType: Integer;  
packSize: LongInt;  
hRex: Fixed;  
vRes: Fixed;  
pixelType: Integer;  
pixelSize: Integer;  
cmpCount: Integer;  
cmpSize: Integer;  
planeBytes: LongInt;  
pmTable: CTabHandle;  
pmReserved: LongInt;  
END;  
PixMapPtr = ^PixelMap;  
PixelMapHandle = ^PixelMapPtr;
```

A PixMap is usually manipulated as a handle. Most of the data structures and routines that work with PixMaps, use or pass handles to the PixMap. For this reason, the 3 high bits of the row width field are used as flags. The high bit of the row width field must be set (1). This indicates that this data structure is a PixMap, not a Bitmap. The next two bits are reserved for future use. For now, they must be unset (0).

Beyond the three normal Bitmap variables, the PixMap has twelve other variables. The pmVersion field contains the version number of Color Quickdraw. Currently this is set to 0. The packType field is used to define the Packing Algorithm that the PixMap uses on the bits. Again, this is normally set to 0 since,

as of now, Color Quickdraw does not support a PixMap packing algorithm. The packSize field contains the number of bytes of the packed image. It is set to 0 if no packing algorithm is used. The hRes and vRes fields contain the horizontal and vertical resolution of the PixMap in pixels per inch. Currently all Mac screens are 72 DPI, thus the hRes and vRes settings of the Pixmap are 72.

The next few fields define the exact layout of the bits in the PixMap's Pixel Image. The pixelType field defines the format. PixelType should be set to 0 to indicate Chunky. A value of 1 indicates "Chunky/Planar" format and a value of 2 indicates "Planar". The first and most important format is the "Chunky" format. In a Chunky Pixel Image, all of a row's pixels are stored consecutively. Thus for a Pixel Image with the depth of 4, the first 4 bits of memory represent the first Pixel.

Originally, Chunky format was the only format that Color Quickdraw could use; all Mac // video cards used "Chunky". However, with the introduction of video cards with more than 8 bits per pixel, the other formats were also used. The Planar and Chunky/Planar formats divide the Pixel Image in memory into separate sections. These sections, called planes, usually represent different color components of a shade. For example, a 24 Bit card might divide the memory up into Red, Green and Blue portions, or components, of memory. Hopefully, this column will be discussing more about 32-Bit Quickdraw in the future. For this article, only Chunky Format will be explored.

For Chunky Format, it is important to realize that for every pixel on the screen there is a group of consecutive bits in memory. This group of bits contains an integer value, representing the color of the pixel being displayed. For example, a PixMap with a depth of 4 would have 4 bits for every pixel in the image. Since 4 bits can contain 16 distinct integer values (0 thru 15), that PixMap could contain 16 different colors at one time. A 8 bit PixMap could contain up to 256 distinct colors.

The pixelSize field of the PixMap defines the depth of the PixMap and it's Pixel Image. The pixelSize must be in powers of 2 for Chunky format (ie. 1, 2, 4, 8). The next field, cmpCount, defines the number of color components (planes). The cmpSize field contains the number of bits per each color component. The planeBytes field gives the offset from one color plane to the next. Since Chunky format has only 1 component, the cmpCount field is set to 1, the cmpSize field should match the pixelSize field, and the planeBytes field should be set to 0 (indicating no offset). Jumping ahead one field, the pmReserved field is exactly that: reserved for future expansion and currently set to 0.

The last field of a PixMap to be explained is the pmTable field. So far, the layout and dimensions of the PixMap have been defined. However, there has been no mention of how the value of a set of bits is converted into a RGB color. Such a conversion is defined in a Color Table data structure. The pmTable field contains a handle to a Color Table data structure.

Color Tables

So far in this article, the terms 'screen' and 'PixMap display' have been used interchangeably. This is not exactly correct. All video display devices are Graphic Devices. Graphic Devices have numerous data structures associated with them, including a

device driver, Color GrafPort, a Color Table and a PixMap. The Graphic Device's PixMap is the entire viewable area of the screen. This is similar to the way traditional Quickdraw has a screenbit variable that is a Bitmap of the entire screen. Other PixMaps that contain portions of the screen must point to the same data as the Graphic Device's PixMap and must share the same Color Table. However that Color Table is owned by the Graphic Device, not any PixMap. Offscreen PixMaps can also be created. These PixMaps would point to a different Pix Image than the Graphic Device. Each PixMap could have it's own Color Table or share a Graphic Device's Color Table. More will be discussed in the next article about the advantages of each method. For now, realize that there is a difference in layout between a Color Table owned by a PixMap and one owned by a Graphic Device.

Both PixMaps or Graphic Devices use the Color Table data structure to convert the bit values of pixels into some RGB color. To do this, a Color Table contains a list of colors, defined in the Color Spec data structure. Each Color Spec contains a RGB color field, defined using the RGBColor data structure which specifies the Red, Green and Blue portions of the color. Each Color Spec also has a number associated with it. Thus if the Color Table has 16 colors, there are 16 numbers associated with it. If the bits that are associated with a specific pixel of a PixelMap are equal to the number of some Color Spec, then that Pixel has that associated RGB value. If the PixMap was part of a Graphic Device, the display would show that RGB color on the Graphic Device. Where the number that is associated with each Color Spec comes from will be explained below.

When some pixels of a PixMap are copied onto another PixMap, each pixel on the source PixMap is converted into the RGB value. Then the closest matching RGB value is found on the destination PixMap. The associated value is then placed in the destination PixMap in the correct bits. Thus PixMaps having various depths and colors can be copied from one and another. While it makes things simpler and faster if the PixMaps share the same Color Table, it is not necessary. Color Quickdraw calculates the correct bit values for all the pixels even if they do not share the same Color Table.

```
RGBColor = RECORD
```

```
  red: Integer;
  green: Integer;
  blue: Integer;
```

```
END;
```

```
ColorSpec = RECORD
```

```
  value: Integer;
  rgb: RGBColor;
```

```
END;
```

```
ColorTable = RECORD
```

```
  ctSeed: LongInt;
  ctFlags: Integer;
  ctSize: INTEGER;
  ctTable: ARRAY [0..0] OF ColorSpec;
```

```
END;
```

```
CTabPtr = ^ColorTable;
```

```
CTabHandle = ^CTabPtr;
```

The Color Table data structure consists of a handle to a variable size record. The first field, the `ctSeed` field, is the version identifier number used internal by Color Quickdraw. It keeps track of changes to the Color Table. Everytime the Table is changed, the `ctSeed` needs to be reset using the Color Manager `GetCTSeed` function. If Color Quickdraw changes the Color Table, it will reset the field. If an application does the changes, it must reset the field. When creating `PixMaps`, this field is set to 0.

The second field of the Color Table is the `ctFlags` field. As mentioned above, some Color Tables are owned by Graphic Devices, while others can be owned by simple offscreen `PixMaps`. If the `PixMap` owns the Color Table, this field must be set to 0. If the Graphic Device owns the `PixMap`, the high bit of the field will always be set. The rest of the bits of the field will be used as flags by the Graphic Device Manager.

The next field of the `PixMap` is the `ctSize` field. It is the number of RGB colors that are defined in the Color Table. The number of colors equals the value of the field minus one, thus a setting of 1 indicates 2 Colors, while a setting of 255 indicates 256 colors. For a Graphic Device, this number needs to be a power of 2. For a `PixMap`, this number can be any positive value. Remember that the bit depth of the `PixMap`, not the `ctSize` field, defines the maximum number of colors for a Pixel Image.

The last portion of the Color Table contains a variable size array of Color Spec data structures. The RGB field contains the exact Red, Green and Blue components of the Color Spec. The number that is associated with that Color Spec is dependent on who owns the Color Table. Graphic Devices use the position of the Color Spec (zero count) in order to find the number. Thus the first Color Spec would have the number 0, the next Color Spec would have the number 1 and so on. In that case, the value field of the Color Spec is used internally by the Color Device manager. Color Tables that are owned by `PixMaps` use the value field to determine what the number of the Color Spec is. Each value field of the Color Table should then have a unique number. While the contents of the value fields can be in any order (first Color Spec, value 9, next Color Spec, value 3, next one, value 23, and so on), it is recommended that `PixMaps` follow the Color Devices method of numbering the Color Specs (first Color Spec, value 0, next Color Spec, value 1, next one, value 2, and so on). This makes it easier to visualize the image.

Graphic Devices have very special rules about the Color Tables associated with them. As mentioned above, the number of colors must be a power of 2, and match the bit depth of the Graphic Device. The first color must be white (RGB value \$FFFF,\$FFFF,\$FFFF) and the last color must be black (RGB value \$0,\$0,\$0). Color Tables used with `PixMaps` are much more unrestricted in their constructions. There can be any number of Color Specs and they may be in any order. Just remember that the `PixMap` that points to the Color Table can only use the first N number of colors where N is dependent on the depth of the `PixImage` (depth 4, 16 colors, depth 8, 256). It is a good rule of thumb to have the first Color Spec be white and the last one be black. Again, this makes it easier to visualize the image.

Icons and Color Icons

Now that an `Image`, a `Bitmap`, a `Pixel Image` and a `PixMap` have been discussed, traditional Icons and Color Icons can be explained. Under Quickdraw, an Icon is a Bit Image of a very specific size. The dimensions of an Icon are always the same. An Icon is a 32 bit by 32 bit black and white image. It's Row width is 4 bytes. Since the dimensions are always the same, the Icon data structure consists of a handle to the Bit Image portion of the Icon.

Under normal Quickdraw, there are two types of Icons; icons stored in 'ICON' resources and icons stored in 'ICN#' resources. The handle of a 'ICON' type Icon contains only the Bit Image. Since the Row Width is 4 bytes and the vertical size is 32, the size of such an Icon handle is 128 bytes. When an 'ICON' type Icon is drawn on the screen, the entire Bit Image is transferred (all 32 by 32 pixels) to the screen.

An 'ICN#' icon's handle contains the Bit Image of the Icon followed by the Bit Mask of the Image. Thus it is twice the size, or 256 bytes. If the Bit Image portion explains what is to be drawn, the Bit Mask portion explains where. Thus each pixel has 2 bits associated with it, an Image bit and a Mask bit. When an 'ICN#' type Icon is drawn on the screen, only the pixels of the Bit Image, whose corresponding Mask bits are set, are then transferred to the screen.

For example, imagine an 'ICN#' type of Icon consisting of a 10 by 10 frame. The Bit Image portion contains a hollow square. The pixels in the frame are set (black). The pixels of the outside of the frame and the inside of the frame are unset (white). The Bit Mask portion of the Icon contains a solid square. The pixels in the 10 by 10 square are all set, while only the pixels outside that 10 by 10 square are unset. When this Icon is drawn on the screen, only the pixels of the Bit Image that are set in the Bit Mask (the ones in the 10 by 10 square) are copied. Thus if the Icon is copied onto a gray background, the frame would be black and the inside of the frame would be white. The idea that an Icon contains a Image and a Mask is an important one in understanding Color Icons.

Given the understanding of all the other data structures, the Color Icon data structure is fairly simple. It contains a `PixMap` defining the Color Image, a `Bitmap` defining the Mask, another `Bitmap` defining the Black and White image, a Handle to the Color Image data and a variable size array of data containing the bitmap images.

```

CIcon = RECORD
    iconPMap: PixMap;
    iconMask: Bitmap;
    iconBMap: Bitmap;
    iconData: Handle;
    iconMaskData: ARRAY[0..0] of Integer;
END;
CIconPtr = ^ CIcon;
CIconHandle = ^ CIconPtr;

```

The first thing to notice is that the Icon can be of any size, not just 32 by 32 pixels. The `iconPMap` field defines a `PixMap` of any given size, depth or color. Second, note that a Color Icon always contains a Mask, held in the `iconMask` field. When a Color Icon

is drawn, only the color pixels in the PixMap which correspond to the set bits in the Mask Bit Image will be copied. Next, notice that the Pixmap will not be drawn on screens with a depth of 1 or 2 (Black and White displays or 4 Color displays). In these cases, the BitMap defined in the iconBMap field would be drawn instead. While the Pixel Image and the Bit Image do not need to look similar, it is better if they do. Both images share the same Mask. Finally, note that while the color Pixel Image data is stored as a handle in the iconData field, the Mask and Bitmap Images are stored at the end of the Color Icon data structure. Thus a Color Icon handle is a variable size handle, depending on the dimensions of the Mask and Bit Image.

Color Icon Resource

Now that the Color Icon data structure has been explored, creating Color Icons can be discussed. While Color Icons can be made with many methods, the most common one is to use a Color Icon Resource. Color Quickdraw has a routine, GetCIcon, that creates a PixMap (and associated Color Table and Pix Image) using a resource template. Color Icon resource use resource type 'cicn'. The important thing to realize is that the resource is used as information in the creation of a Color Icon. Once the Color Icon is created, the template is released by the call. This is different than other traditional Quickdraw resources (ie. 'ICON' or 'ICN#') where the resource is the entire data structure. The 'cicn' resource is a merge of the various data structures and fields that the Color Icon data structure contains. Thus there are similar fields and values. The 'cicn' resource format and content is as follows:

Name Date Type Use

Icon PixMap Portion

baseAddr	Handle	0
rowbytes	Integer	rowbytes of PixMap
bounds	Rect	boundary rectangle of PixMap
pmVersion	Integer	0
packType	Integer	0
packSize	LongInt	0
hRes	Fixed	72
vRes	Fixed	72
pixelType	Integer	0 (Chunky)
pixelSize	Integer	bits per pixel of PixMap
cmpCount	Integer	1
cmpSize	Integer	bits per pixel of PixMap
planeByte	LongInt	0
pmTable	Handle	0
pmReserved	LongInt	0

Icon Mask Portion

baseAddr	Handle	0
rowbytes	Integer	rowbytes of Mask
bounds	Rect	boundary rectangle of Mask

Icon Bitmap Portion

baseAddr	Handle	0
rowbytes	Integer	rowbytes of BitMap
bounds	Rect	boundary rectangle of BitMap

Icon Data	Handle	0
-----------	--------	---

Mask Data	Mask's rowbytes * Mask's vertical size
-----------	--

Bitmap Data	Bitmap's rowbytes * Bitmap's vertical size
-------------	--

Color Table Portion

ctSeed	LongInt	0
ctFlag	Integer	0
ctSize	Integer	Number of Color Specs - 1
ctSpecs	Number Specs*8 array of Color Specs	

PixMap Data	PixMap's rowbytes * PixMap's vertical size
-------------	--

Many of the fields are place holders. As such, they are set to 0. Remember that Integers are 2 bytes in size, Handles are 4 bytes, LongInt are 4 bytes, Fixed (real values) are 4 bytes and Rect (Rectangles) are 8 bytes (4 Integers). The rest of the field directly related to the Color Icon data structure.

Color Icons in Menus and Dialogs

The two most common usages of normal Icons are placing them in Menus and drawing them in Dialogs. In both cases, the 'ICON' resources number is somehow defined in the Menu and Dialog. The Menu and Dialog Manager then displays the Icon.

The method to place icons in the menu is slightly complicated. If a Menu Item is created using the GetMenu or AppendMenu procedures that contains a circumflex (^) followed by an ASCII character beyond 48, then the ASCII value of the character subtracted by 48 then added to 256 gives the resource number of the Icon to be displayed in that Menu Item. This strange numbering scheme means that a Menu Item with '^1' in it will display the Icon with the resource number of 257. '^2' will give resource number 258, while '^9' will give 256. Also the SetItemIcon procedure can be used to directly attach a Icon to a Menu Item. In that case, 256 must be added to the Icon number in the procedure to find the correct ICON resource number. Thus Icons that are displayed in the Menu must be in the range of 257 to 512.

To display Icons in Dialogs and Alerts, the Dialog Item List must contain an iconItem. That Item will contain the Display rectangle in local coordinates of the Dialog and the 2 byte resource ID. Any ICON resource number can be used.

With the implementation of Color Quickdraw, changes to the Menu Manager and Dialog Manager have been added. Now, when an Icon is referred to by a Menu or Dialog, first the resource number is calculated. Next, a Color Icon resource ('cicn') with

that resource number is looked for. If that resource exists, a Color Icon is created using it. That Color Icon will be displayed in the Menu or Dialog. If a 'cicn' resource with that resource number does not exist, the 'ICON' resource with that number is looked for and used.

This method of using 'cicn' resource, when one exists, was well thought out. Imagine a program is created that has Icons in the menu and in it's dialogs. The program is designed to run on color and non-color Macintoshes. For each Icon needed, an ICON resource and a 'cicn' resource are created with the same number. When the program runs on non-color Macintoshes, the 'ICON' resource is loaded into the computer and displayed. When the program is run on a color Macintosh, the 'cicn' resource is found and displayed. Color Icons can be added to a program without any source code changes. Some of the Colorization programs available use this idea to add color to programs already created.

Creating Color Icon Resources

The Color Icon resource structure is a complex one. It is not a fixed width structure. Many of the field sizes are dependent on the settings of other fields. For this reason, most resource tools (ResEdit, RMaker) are not capable of easily manipulating the resource structure. The only resource tool that can do this is MPW's Rez tool. While MPW does have a 'cicn' resource template that works, it has a couple of faults. First of all, it is hard to imagine the image that the Color Icon is trying to create. Changing and editing the Color Icon is difficult. Secondly the template does not do a good job of deciphering the Color Table portion of the data structure. Lastly, many of the fields of the data structure could be precalculated or predefined.

To solve these problems, a modified 'cicn' resource template has been created (see resource code example #1). First a couple of assumptions must be made. The Color Icon that is to be created must be within a 32 by 32 pixel size. Since the standard size of the tradition Icon is 32 by 32, this is usually acceptable. Secondly, the depth of the Color Icon is 4 bits or 16 colors. For most uses, 16 colors are enough.

Given these rules, large portions of the resource data structure can be precalculated. Many of the fields are preset, while other variable size fields can be defined as a fixed size. More importantly, the Pixel Image and Bit Images can be displayed in Hex and Binary, respectively, so that an image can be more easily seen. Examine resource code example #2. Five different Color icons are displayed in a more viewable format. Resource code example #2 also contains the resource code for an Alert that displays the icons. Creating Color Icons this way is much easier than using the template provided by MPW! Code example #3 gives the Pascal and Resource source for a very simple demonstration application. The application, cicnFun, displays the Color Icons in an Alert. With this, an user can quickly see the Color Icons that were created.

Listing #1: CIcon.r

```
/*
 * File CIcon.r
```

```

*
* Created by Steve Sheets for MacTutor
*
* revised cicn Resource Template
*
*/

type 'cicn' (
    longint = 0;          /* Base address */
    integer = $8010;      /* New pixMap flag & Offset
tonext row*/
    integer = 0;          /* Bitmap bounds */
    integer = 0;
    integer = 32;
    integer = 32;
    integer = 0;          /* pixMap vers number */
    integer = 0;          /* Packing format */
    longint = 0;          /* Size of packed pixel data*/
    unsigned hex longint = $00480000; /* h. resolution
tion (ppi) (fixed) */
    unsigned hex longint = $00480000; /* v. resolution
tion (ppi) (fixed) */
    integer = 0;          /* Pixel storage format */
    integer = 4;          /* # bits in pixel */
    integer = 1;          /* # components in pixel*/
    integer = 4;          /* # bits per field */
    longint = 0;          /* Offset to next plane */
    longint = 0;          /* Offset to color table*/
    longint = 0;          /* Reserved */

    /* IconMask (bitMap) record */
    fill long;            /* Base address */
    integer = 4;          /* Row bytes */
    integer = 0;          /* Bitmap bounds */
    integer = 0;
    integer = 32;
    integer = 32;

    /* IconBMap (bitMap) record */
    fill long;            /* Base address */
    integer = 4;          /* Row bytes */
    integer = 0;          /* Bitmap bounds */
    integer = 0;
    integer = 32;
    integer = 32;

    longint = 0;          /* Handle placeholder */

    array [32] (          /* Mask Data */
        unsigned binary longint;
    );

    array [32] (          /* BMap Data */
        unsigned binary longint;
    );

    /* PMapCTab */
    longint = 0;          /* ctSeed */
    integer = 0;          /* transIndex */
    integer = $Countof(ColorSpec) - 1; /* ctSize*/
    wide array ColorSpec (
        hex integer;      /* value */
        unsigned integer; /* RGB: red */
        unsigned integer; /* green */
        unsigned integer; /* blue */
    );

    array [32] (          /*PMap Data */
        hex string[16];
    );
);

Listing #2: cicnFunRes.r
```


[illegible]

```
( 0b0000000000000000 111111110000000000,
  0b0000000000000000 111111110000000000,
  0b0000000000000000 100110010000000000,
  0b0000000000000000 100110010000000000,
  0b0000000000000000 1111111100000001001,
  0b0000000000000000 1111111100000001001,
  0b0000000000000000 000000010000000000 111,
  0b00011000000000 11111111000000 111,
  0b00000100000000 11111111100000 110,
  0b000001000000 110000 1100110111000110,
  0b00011110001100011000111011100,
  0b000000 11011100001100011111000,
  0b00000001111000001100001110000,
  0b000000001110000011100000100000)
```


499

[illegible]

```
(
"$00000000000005555555555550000000",
"$00000000000005000000000000000000",
"$00000000000005000022222000500000",
"$00000000000005000000000020050000",
"$00000000000005000000000020050000",
"$00003000000050000000000020050000",
"$00003000000050000000000020050000",
"$00030000000050000000000020050000",
"$300300030000050000000000020050000",
"$30000003000005000022222000500000",
"$300000000000050000000000000050000",
"$300000000000050000000000000050000",
"$0300303000000000000000555550050000",
"$00330300000000000000000000050000",
"$0000000044444000000000000050000",
"$00000044000000000000000000000000",
"$00000400000000055555555555550000",
"$00000400000000000000000000005000",
"$00000044444000000000111111100500",
"$0000000000400000000010101010050",
"$000000000040000000000010101005",
"$0000000000400000000000000000055",
"$00000000004000000000000055555000",
"$00000000004400000000000000000000",
"$00000000000040000000000000000000",
"$00000000000000000005500000000000",
"$00000000000000005005000000000000",
"$00000000000000500200500000000000",
"$00000000000000002000500000000000",
"$00000000000000000000500000000000",
"$00000000000000000000500000000000"
);
```

Pascal Procedures

Writing INITs in Pascal



Steve Kiene
Lincoln, NE
Volume 5, Number 10

Writing INITs in Pascal.

Why INITs: Trap Patching, etc...

INITs are pretty hot items these days. I have at least six in my System Folder and I know people that have fifteen or more. INITs do anything from making it easier to move through SFGetFiles (SFScrollINIT by Andy Hertzfeld) to creating large virtual desktops (SteppingOUT II). INITs are used to customize the Macintosh. They are very simple to install and remove (dragging them into or out of the System Folder).

The majority of INITs trap patches. Patch trapping is simply the rewriting of the Macintosh's internal routines or the modification of these routines for special situations. This is done by using the traps GetTrapAddress, SetTrapAddress, NSetTrapAddress, and NGetTrapAddress. Usually the programmer calls GetTrapAddress and saves the address for later use; the programmer then puts this address somewhere, usually in his/her code so that his/her code can check for the special situation, do whatever—taken that it is the special situation, and then call the original trap address. **All trap patches should call the original trap address;** this allows traps to be patched more than one time and is the best bet for compatibility insurance.

I would guess that 90% of all INITs are written in assembly. I'm sure they probably say that INITs are much easier to write in assembly. Well, it's probably true, if you know assembly, but, if you don't know assembly, I would say that it isn't all that easy. This article, since it is in the Pascal section, demonstrates how to write an INIT in Pascal, includes full source to an INIT that allows the cursor to wrap around the screen, and gives a couple of hints on writing INITs in general.

Having the INIT install our patch.

INITs must load their code into the system heap so that the code will stay around as application's are launched (the application heap is flushed every launch or return to the Finder). The INIT part of the code should put the trap patch or whatever code into the system heap. In CursorWrap, the included INIT, we create a pointer in the system heap that is four bytes greater than the size of our VBL task code; we then BlockMove the code into the pointer starting at the location of the pointer plus four. We store the pointer to our VBL task in the four bytes before our code so that we have easy access to it. The code is written in MPW Pascal, but should be easy to convert to any other development system.

```
Unit CursorWrap;
Interface
Uses
  ($LOAD_MAC.Dump)
  MemTypes, QuickDraw, OSIntf, ToolIntf, PackIntf, Script;
```

```
Procedure SetUpVBL;
Procedure Wrap;
```

```
Implementation
Procedure SetUpVBL;
```

```
var
  theVBL      : VBLTask;
  myQElem     : QElemPtr;
  myErr       : OSErr;
  SaveZone    : THz;
  SizeNeeded  : Longint;
  PatchPtr    : Ptr;
  theCode     : Handle;
  thePtr      : ^LongInt;
  dummyErr    : OSErr;
Begin
  (* Get handle to our code *)
  theCode:=GetResource('INIT', 1);
  (* Use the system's Heap *)
  SaveZone:=GetZone;
  SetZone(SystemZone);
  (* Get size of our patch code and our QElem ptr *)
  SizeNeeded:=SizeResource(theCode)-(LongInt(@SetUpVBL)-
    LongInt(theCode))+sizeof(QElem);
  ResrvMem(SizeNeeded);
  If MemError<>NoErr then
    begin
      (* If not enough room in system heap then get out *)
      SysBeep(1);
      SetZone(SaveZone);
      exit(SetUpVBL);
    end;
  (* Get new ptr for our code *)
  PatchPtr:=NewPtr(SizeNeeded+4);
  (* blockmove our code in *)
  BlockMove(@Wrap, Pointer(Ord(PatchPtr)+4), SizeNeeded);
  (* get new ptr for our VBL task *)
  myQElem:=QElemPtr(NewPtr(sizeof(QElem)));
  (* restore zone *)
  SetZone(SaveZone);
  (* put our vbl task ptr's address into the ptr where our
  patch code will be *)
  thePtr:=Pointer(PatchPtr);
  thePtr^:=LongInt(myQElem);
  (* set up VBL and install *)
  with theVBL do
    begin
      qType:=Ord(vType);
      vblAddr:=Pointer(Ord(PatchPtr)+4);
      vblCount:=6;
      vblPhase:=0;
    end;
  myQElem^.vblQElem:=theVBL;
  dummyErr:=VInstall(myQElem);
End;
(-----)
Procedure Wrap;
(**
This code allows the cursor to seemly wrap around the screen
when the <Option> key is held down. It checks the low level
interrupt mouse location (mTemp) and if it is on one edge of
the screen it moves it to the the other edge. It puts the new
cursor coordinatate into mTemp and then puts $FF into cursor-
New-this tells the cursor routines that the cursor has moved...
```

```

**)
const
  CurrentA5 = $904;
var
  myQElem      : QElemPtr;
  thePtr       : ^LongInt;
  CursorCoordPtr : ^Point;
  ChangedPtr   : ^Byte;
  changed      : Boolean;
  theMap       : KeyMap;
  currGDevice  : GDHandle;
  mouseRect    : Rect;
  myRectPtr    : ^Rect;
  myPtr, myPtr2 : ^LongInt;
  myRect       : Rect;
Begin
  ( * Get the keymap and check if option down; if is not then
do not allow wrap * )
  GetKeys(theMap);
  If theMap[58] then
    Begin
      ( * set up ptr to low memory global of cursor location * )
      CursorCoordPtr:=Pointer($828);
      changed:=false;
      ( * get rectangle of screen * )
      currGDevice:=GetGDevice;
      If currGDevice<>Nil then
        mouseRect:=currGDevice^.gdRect
      else
        begin
          ( * Use currentA5 to get offset to
screenBits.bounds * )
          myPtr:=pointer(CurrentA5);
          myPtr2:=pointer(myPtr^);
          myRectPtr:=Pointer(myPtr2^-116);
          mouseRect:=myRectPtr^;
        end;
      InsetRect(mouseRect,1,1);
      ( * check cursor location and change it if wrapping * )
      With CursorCoordPtr^,mouseRect do
        Begin
          if v<=top then
            Begin
              v:=bottom-1;
              changed:=true;
            End
          else if v>=bottom then
            Begin
              v:=top+1;
              changed:=true;
            End;
          if h<=left then
            Begin
              h:=right-1;
              changed:=true;
            End
          else if h>=right then
            Begin
              h:=left+1;
              changed:=true;
            End;
          End;
          ( * if we changed the cursor location then set the low
memory global cursorNew * )
          If changed then
            Begin
              changedPtr:=Pointer($8CE);
              ( * this tells the cursor drawing routines that the
cursor is moved * )
              changedPtr^:=$FF;
            End;
          End;
          ( * get ptr to VBL taks from where we originally put it * )
          thePtr:=Pointer(Ord(@Wrap)-4);

```

```

( * reset the vblcount so that we are executed again * )
myQElem:=QElemPtr(thePtr^);
myQElem^.vblQElem.vblCount:=6;
End;
End.

```

Another Way

There is another way to store variables. In our example INIT this would be the VBL pointer address. This way involves using a header file that is written in assembler. The header file does nothing but branch over instructions that take up space. The header would look something like the following in MPW assembly:

```
Header  MAIN EXPORT
```

```
Header1
  BRA.S 01
  DC.L 0
  DC.L 0
  DC.L 0
01
  END

```

This header would leave room for twelve bytes, three handles, or whatever. To use this you would have to link the Header.a.o in first and reference it in your pascal code. A little help:

```

Procedure Header; EXTERNAL; (must be after Implementation )
myPtr:=Ptr(Ord(@Header)+2); ( myPtr points to the first byte
of storage )

```

The above statement returns a pointer to the address of the header plus two bytes, so that we skip over the branch statement.

You would use this type of code when you are writing a WDEF, CDEF, etc. and you want to have some type of communication between your patches and your custom definition. We used this type of code when we wrote Tear-Off Menus and wanted our custom WDEF and MDEF to be able to access globals that our Menu and Window Manager traps used.

Added tips for coping with problems.

When writing INITs that are a little heavier than a 'one trap patch' INIT or a simple VBL installation, then you are most likely going to run into problems with certain programs. Maybe it will be because the guys at Microsoft didn't quite follow all of the rules or perhaps it's some other reason. Regardless the problem, the answer is to not install when you know you are not compatible. The easiest way to do this is to try and load the resource type of the creator type (ie. MSWD for Microsoft Word) when the application is launched.

```
theType:=GetIResource('MSWD',0);
```

```
If theType<>Nil then ( the application exists )
```

Since some users can and do rename their applications, one can not just check for application names. Creator types should be unique (if they are registered with Apple) and no problems should be encountered this way.

Good luck with your INITs.



Pascal Workshop

Dispatch Aliasing



*Roger Horton
Winston-Salem, NC
Volume 5, Number 11*

Dispatcher - An Application Aliasing and Launching Utility

[Roger Horton is a Research Adjunct Assistant Professor at the Medical School of Wake Forest University, and a private consultant. He is currently in a masters degree program in computer science at the University of North Carolina, and has been programming in the Macintosh environment since 1985.]

Recently, I completed a prototype of a database application for a dental office specializing in the treatment of facial pain. I choose 4th Dimension as the development environment due to its flexibility in reporting and for the ease with which external code routines can be incorporated. However, 4D has the habit of creating a large number of database files (for indexes, resources, data definitions, etc.), which can be very unsettling to the typical office user who happens to take a glimpse inside a 4D application folder. When the users reviewed the prototype they pointed this out, and requested that they be spared the intimidation of wondering what all the files were for. At this point I began to consider the options for simplifying the process of installing and starting the application.

Avoiding Confusion

A frequent point of confusion for users of custom database applications is setting up and running the application. Something as simple as starting an application can be difficult for novice Mac users, if they have to traverse too many volumes and folders. In other text oriented operating systems, developers could rely on shell scripts to simplify a user's interaction with the file system. On the Mac, macro programs have sprung up to fill this void. These can help, but often add more complexity than they eliminate.

If the designer can personally walk each new user through the process of installation and startup, all is well. But all too often, the system is shipped to distant users who rely primarily on the documentation. As most of us have discovered, the knowledge level of the user varies widely, and the phone time spent in supporting these users, especially in the first weeks of use, can be considerable.

My goal was to keep the user out of the 4D application folder, with its many files, and allow a launch of the program from the Finder, using a single, easily identifiable icon. Finder alternatives were not considered, due to the user's familiarity with the Finder and our desire to use Multifinder for coordination of other programs with the database application. I reviewed some macro processors that could potentially automate the launch process, but generally found that they were sensitive to the initial state of the desktop and were likely to be confusing to the average user. Since I hoped to simplify, instead of complicate, the training

process, I considered a custom solution. My initial thoughts centered around writing a small utility program that would take care of locating and launching the database application. I had previously written code to launch one application from another and so I set about considering how to use this code as a basis for a custom launcher application. I should note that normally I would have developed this type of utility program in C, but found that MPW version 2.0.2 C did not support the in-line launch code (but does in version 3.0). This caused me to turn to MPW Pascal, although I tried to develop the code for easy translation to C (avoiding booleans, 'with' statements, etc.).

Help from the Finder

My first intention was to build a launcher utility program that could be placed outside the 4D database folder. It would know (or could be shown) the location of the database folder, and could set up the Finder information record such that 4D would open the database after it was launched. This required storing the location of the 4D folder in the launcher program's data or resource fork. This appeared to me to be a workable, but inflexible approach to the problem. There was also the problem of storing data in a resource fork, which Apple now discourages due to lack of access control to resource forks on networked systems.

A second approach was to rely on some help from the Finder. Using information from the Desktop file, the Finder can launch an application when the user opens (or prints) one of its documents. By taking advantage of this ability, I could create a dummy document and set its filetype to the filetype of the launcher program. It could then be placed anywhere in the file system (directly on the desktop, for example) and still initiate the launch program without knowing where the program file was. No file system paths need be remembered, but the launch program would always have to reside in the same folder as the application to be launched, and information about the target application would have to be imbedded in the program code.

Generalizing the Dispatcher

Any good computer scientist would realize that up to this point my approach was a very narrow solution to the problem, and lacked the generality we all strive for. So, I chose a solution that allowed for greater flexibility in setting the launch parameters. To do this, I relied on the dummy launch document to store the launch parameters, rather than embedding them in the program code or in the resource fork of the launch program. By doing so, the dummy documents (hereafter called launch settings files) become a sort of alias for the program they cause to be launched. They also store additional information needed for setting up the launch of the target application. Using this solution, many

settings files could be created, each capable of launching a different application. These settings files could then reside anywhere in the file system, with a single program acting as the dispatcher of applications, according to the data contained in each separate launch settings file. Hence the name “dispatcher”.

Remembering the Launch Parameters

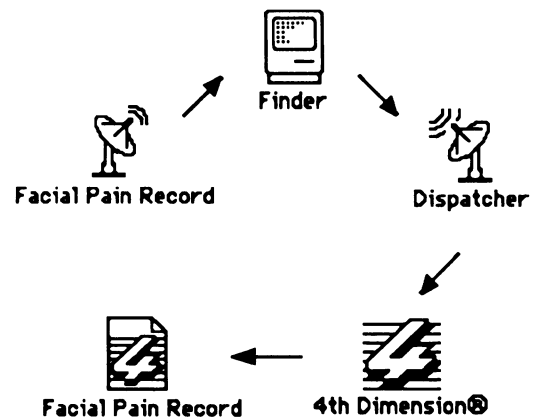
To launch a target application, and tie in an associated launch document, the dispatcher program stores two data records in each launch settings file. The two records are the same, each containing the name of the file, the directory ID of its parent folder, and the name of the volume containing the directory (if the file is at the volume root level, the directory number is 2 and the directory name is the same as the volume name). Some additional information is also stored with each of the two records to help with the launch. The filetype is remembered, and can be used when it's necessary to ask the user to help find a file that has changed location. Also as part of the data, a flag is stored that determines whether or not the target application will be told to open the document after it is launched.

Given that multiple dispatcher settings files can be present, it has to be easy to initialize and update the data in each one. To permit this, the dispatcher program tests the command key before launching the target application. If the key is down, a dialog window is presented that allows the user to alter the previous settings directly, or to change the settings by searching for the application and its associated document using the standard file dialog. It is also possible to set or unset the flag that determines whether the application should be instructed to open the associated document.

About Launching

Apple Technical Support has provided detailed code examples illustrating how to launch one program from another (in tech notes 52 & 126). I've based my code on these notes, including the recommendation not to use sublaunches. Avoiding sublaunches was not a significant limitation in my particular application, since there was no need to regain control after the launched application finished. In the event that you do use the sublaunch feature, it's interesting to note that the Launch routine is now a function that returns a result when run under Multifinder. You can test the result of the launch and inform the user of errors. However, it turns out that a sublaunch, rather than a launch, must be performed in order to have Multifinder return a result code that can be processed by the sublaunching program.

The overall process of launching an application with the dispatcher requires a number of steps (Figure 1). The process begins with the user opening (by double-click or menu) the dispatcher settings file. This causes the Finder to load the data document's working directory reference number (as well as other information) into the Finder information record referenced by the AppParmHandle global, and then launch the dispatcher. When the dispatcher gets control, it uses GetAppFiles to obtain the reference number of the settings file. This number is then used to locate and open the file for reading/writing. The two data records, one for the target application and one for the associated docu-



Launch Sequence

- 1) User double clicks on settings file
- 2) Finder launches Dispatcher
- 3) Dispatcher launches target application
- 4) Application opens associated document

Figure 1

ment, are read in. If the dispatcher can open the file, but can't read any records, it assumes the data has not been initialized, and sets up some initial dummy data. Normally this is not necessary, since new data documents can be created by duplicating existing documents and then editing the existing settings. However, when setting up the dispatcher for the first time, a file utility needs to be used to create the first data document. Remember to set the filetype to 'LCHD', the dispatcher's settings file filetype (the dispatcher's filetype is LCHA).

Having read in the necessary launch parameters, the dispatcher must now take care of some housekeeping related to the Finder information that will be passed on to the target application. First the AppParmHandle is obtained from the system global variable at memory location \$AEC. The current heap zone is set to the system heap, and the existing handle is disposed. A new handle is allocated in the system heap allowing sufficient space for the information related to a single target application document. Then the current heap pointer is set back to the application's heap. I can recall the admonition in one of Apple's early Macintosh training courses to avoid allocating anything in the cramped system heap of the 128K and 512K Mac's. Fortunately, Apple now provides a larger, variable size system heap and looks more favorably (I think) on developers allocating structures there.

With a new Finder information record allocated, the launch data is loaded into records corresponding byte-for-byte to the Finder information records, and the records are copied to the system heap at the location referenced by the newly allocated handle. This handle is then loaded into the system global at location \$AEC. Be sure to note that if the target application is to be launched without an associated document, then the Finder information must be zeroed out correctly, since the Finder will

not be intervening to perform its usual housekeeping. With the application parameters correctly loaded, all that is left to do is to set the default working directory to the target application's directory and call the launch routine.

Navigating the HFS

Something should be said about the way the dispatcher navigates the hierarchical file system. After the target application and document are initially located, the dispatcher must assure that enough information is saved to permit them to be located on subsequent invocations.

To adequately specify a file to the file system, a number of options are available. A full pathname from the root volume is one possibility. However, tech note #238 discourages this, since the file's directory may be moved, making the pathname invalid (also note that system 7.0 will provide unique 32 bit file ID numbers). For current system versions, the dispatcher remembers the filename, directory ID number, and the volume name, and uses them together to locate the target file. Since multiple volumes may be mounted, the program checks the stored volume name against each mounted volume for a match. It is possible that two volumes might have the same name and confuse the dispatcher, but this is uncommon and left as an enhancement to the program. Having located the volume, the program tests for the presence of the target file by using the filename and directory ID number. This process is carried out for both target files (application and associated document) and, if successful, is followed by the opening of working directories for both files.

A number of errors can occur while trying to locate the target files. Most commonly, one or both of the files might have been moved to another directory and/or volume. If this occurs, the program responds by asking the user to find the missing files, using the standard file dialog. As each file is found, the volume name, directory ID, and filename are saved so as to be available during the next use of the dispatcher. After the correct location information is obtained, it is written back to the settings file prior to launching the target application. If any unrecoverable errors occur, or the users cancels any of the dialogs, the dispatcher terminates without carrying out the launch.

One other point that should be made comes from personal experience with the dangers of using PBHSetVol. Anyone considering using the low level file manager calls should make sure they read tech note #140. Normally, most file manager calls will accept a working directory reference number in place of a volume reference number. This maintains compatibility between the flat and hierarchical file systems. Unlike other calls though, PBHSetVol has the habit of taking a working directory reference number in ioVRefNum and then setting the default directory back to the root. This can cause more than a few minutes of confusion trying to find out why files suddenly can't be found.

Using and Extending the Dispatcher

The dispatcher program has been written to require minimal user intervention. Ideally, a typical user should not even know that an intermediate program has been part of the launch of the target application. However, when initializing the file location

data, or accommodating the movement of a file to a new location, the dispatcher's settings dialog must be used (Figure 2). Its occasionally useful to view the dialog just to confirm what the settings are, especially if the settings haven't been used in a while. The information presented in the dialog is very simple. Two edittext items are present to allow the user to enter the name of the target application and associated document. A check box item is also present that lets the user specify whether the application should be launched with or without the associated document.

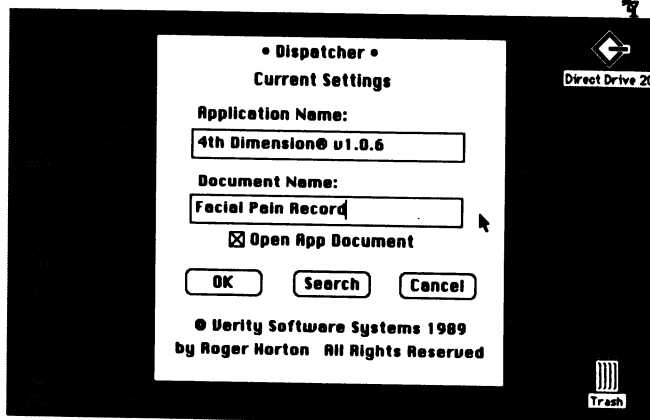


Figure 2

A second method to specify the target file data is also provided. A "Search" button is present that, when pressed, sets the filenames to "XXXXXXXX" and goes directly to the standard file dialogs to locate the files. This method completely clears the previous filenames, and normally is used when new settings are being created.

As mentioned above, the dispatcher program and data documents can be located anywhere in the file system. This has been especially handy for me when using Multifinder. I'm sure that any Multifinder user is familiar with the desktop clutter that occurs when more than one application is open. Having a lot of Finder folders open on the desktop only adds to this clutter. My solution is to keep a set of dispatcher settings files directly on the desktop corresponding to the projects on which I'm currently working (Figure 3). All other folders are kept closed. This makes for much less clutter, and saves me the trouble of opening and closing folders as I launch each application.

In my case, the original purpose of the dispatcher was to make installation and launching easier for users of 4th Dimension custom applications. My database application is supplied with the dispatcher located in a folder amongst the other 4D files. The settings file is located outside this folder. Installation simply involves copying the settings file and database folder onto the user's hard disk. The user never has to look inside the database folder, except from the standard file dialog when first installing the program. If the volume on which the application is to be installed happens to be known ahead of time, then its possible to completely preconfigure the settings file.

Its easy to conceive of some enhancements to the dispatcher that would make it more useful. Options could easily be added to

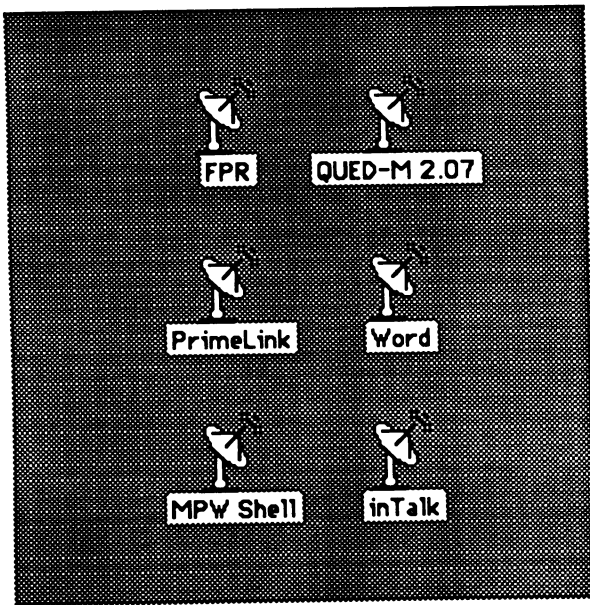


Figure 3

allow more than one data document to be associated with the target application. In this way, an application could be opened with a number of associated documents, similar (although much more limited) to a user startup script in the MPW environment. Provision could also be made for retention of the location of the dispatcher program in the data documents. This would make it possible to use the dispatcher with Finder substitutes. The dispatcher could also be made to create its own settings files, making initial use of the program simpler. Its very likely that future developments in the Mac operating systems (real file aliasing, etc.) will help provide for some of the features of the dispatcher program. But for the time being, it provides users with an intuitive method of launching custom database applications. Beyond that, it can help to reduce desktop clutter while providing a general method for launching any application with an associated document.

Listing: dispatcher.make

```
dispatcher      ff      dispatcher.r
Rez dispatcher.r -o dispatcher -append
SetFile -a B dispatcher -c LCHA -t APPL
dispatcher      ff      dispatcher.p.o
Link dispatcher.p.o @
  "(Libraries)"Interface.o @
  "(Libraries)"Runtime.o @
  "(PLibraries)"Paslib.o @
  -o dispatcher
dispatcher.p.o  f      dispatcher.p
Pascal dispatcher.p
```

Listing: Dispatcher.p

```
(
File:      dispatcher.p
Program:   dispatcher
Author:    Roger A. Horton
           Verity Software Systems
```

Copyright 1989, All Rights Reserved

Creation: 6/4/89

Purpose: Create a small utility program to launch applications

from the Finder with associated launch documents. This is especially useful for custom applications that require a number of files, and that may cause a user confusion in installing and launching the application.

Technique: This program performs a launch (not a sublaunch) using the code of Mac Tech Note #126 as a model. The utility does not directly initiate the launch, but relies on associated

data documents to initiate the process, and to determine which application and documents are to be launched. The launch utility and document(s) can be located anywhere in the file system. By double clicking on a launch document, the user causes the Finder to locate and launch the launch utility program. The utility then reads the launch data document to obtain the name and location of the application and its associated document. The Finder application parameters are then set up, and the application is launched. If the location of the application or its associated documents has changed since the last launch, the user is prompted to find their new location. In essence, the launch utility documents act as aliases for target applications, and add additional usefulness by providing for automatic control of associated application documents.

Program History:

)

program scan;

```
($D+)      ( Macsbug/TMON symbols )
($R-)      ( Turn off range checking )
```

uses

```
( Macintosh toolbox units )
PasLibIntf, MemTypes, QuickDraw, OSIntf, ToolIntf, PackIntf,
SANE, PrintTraps, ROMDefs
( Custom units )
```

const

```
active = 0;      ( for use in controls and menus )
inactive = 255;
```

```
DfltDialogID = 200; ( resource ID for config dialog )
```

```
StopDialogID = 201; ( resource ID for stop alert dialog )
```

```
AppParmGVar = $AEC; ( AppParmHandle sys global variable )
```

type

```
( general types )
```

```
pLaunchStruct = ^LaunchStruct; ( Launch record )
```

```
LaunchStruct = record
```

```
  pName: ^Str255;      ( pointer to application name )
  param: integer;      ( alternate video, audio buffers )
  LC: packed array[0..1] of char; ( extended parameters )
  extBlockLen: longint; ( extra block length )
  fFlags: integer;      ( finder file info flags )
  launchFlags: longint; ( bits 30,31 = 1 for sublaunch )
end;
```

```
FileInfo = record      ( Finder files )
```

```
  vRefNum: integer;      ( 2 bytes )
  fileType: OSType;      ( file type - 4 bytes )
  version: boolean;      ( 1 byte )
  unused: boolean;      ( 1 byte )
  fName: Str255; ( 1 length byte, variable length string )
end;
```

```
APHdl = ^APPtr; ( Finder info )
```

```
APPtr = ^APRec;
```

```
APRec = record
```

```
  message: integer;      ( 2 bytes )
  count: integer;        ( 2 bytes )
  files: array[1..1] of FileInfo;
```



```

end;
LaunchData = record (application & document data settings)
  vName: Str255;    ( volume name )
  vRefNum: integer; ( working directory reference # )
  dirID: longint;   ( directory ID # )
  fName: Str255;    ( file name )
  fType: OSType;    ( file type )
  useIt: integer;   ( whether to open associated document )
end;

var
  ( file system parameter blocks )
  myPB: ParamBlockRec;
  myHPB: HParamBlockRec;
  myInfoPB: CInfoPBRec;
  myWDPB: WDPBRec;
  ( launch data records )
  applData: LaunchData; ( application )
  docLData: LaunchData; ( document )
  ( miscellaneous )
  done: integer; ( done flag )
  lcnt: integer; ( loop counter )

( ***** Utility functions and procedures ***** )
procedure AlertMsg(msgStr1, msgStr2:Str255);
( Procedure to print alert message.
  To print numerics, use something like this:
  testStr1, testStr2: Str255;
  NumToString(reqCnt, testStr1);
  NumToString(reqCnt, testStr2);
  AlertMsg(testStr1, testStr2); )
var
  theRect:rect;
  ignore:integer;
begin
  ParamText(msgStr1, msgStr2, '', '');
  ignore := StopAlert(StopDialogID,NIL);
end; ( procedure AlertMsg )

( ***** Error Handling ***** )

function GetErrMsg(result:OSErr):string;
begin
  result := abs(result);
  case result of
    18: GetErrMsg := 'driver error during status operation';
    19: GetErrMsg := 'driver error during read operation';
    20: GetErrMsg := 'driver error during write operation';
    27: GetErrMsg := 'driver I/O error caused abort of
operation';
    28: GetErrMsg := 'driver not open';
    33: GetErrMsg := 'the file directory is full';
    34: GetErrMsg := 'all allocation blocks on the volume
are full';
    35: GetErrMsg := 'the specified volume is not mounted';
    36: GetErrMsg := 'there was an unspecified I/O error';
    37: GetErrMsg := 'the file or volume name is bad';
    39: GetErrMsg := 'logical EOF reached unexpectedly';
    40: GetErrMsg := 'attempt made to position before start
of file';
    42: GetErrMsg := 'too many files are open';
    43: GetErrMsg := 'the file could not be found';
    44: GetErrMsg := 'the volume is locked by hardware
setting';
    45: GetErrMsg := 'the file is locked';
    46: GetErrMsg := 'the volume is locked by a software
flag';
    47: GetErrMsg := 'the file is already in use';
    48: GetErrMsg := 'a file with specified name exists';
    49: GetErrMsg := 'the file is already open for read/
write';
    50: GetErrMsg := 'no volume specified and no default
volume';
    51: GetErrMsg := 'a non-existent path was specified';
    52: GetErrMsg := 'the was an error finding current

```

```

position in file';
    53: GetErrMsg := 'the specified volume is not on-line';
    54: GetErrMsg := 'attempt to open a locked file for
writing';
    55: GetErrMsg := 'attempt to mount an already mounted
volume';
    56: GetErrMsg := 'the specified drive number is not
mounted';
    57: GetErrMsg := 'the volume lacks a Macintosh format
directory';
    58: GetErrMsg := 'there was an external file system
error';
    59: GetErrMsg := 'there was a problem during renaming';
    60: GetErrMsg := 'the master directory block is bad';
    61: GetErrMsg := 'read/write permission does not allow
writing';
    108: GetErrMsg := 'there is insufficient application
memory';
    109: GetErrMsg := 'a nil master pointer has been
encountered';
    111: GetErrMsg := 'attempt to operate on a free block';
    112: GetErrMsg := 'attempt to purge a locked block';
    117: GetErrMsg := 'the block is locked';
    120: GetErrMsg := 'the directory could not be found';
    121: GetErrMsg := 'too many working directories are
open';
    122: GetErrMsg := 'a folder cannot be placed in its own
subfolder';
    123: GetErrMsg := 'attempt HFS operations on non-HFS
volume';
    127: GetErrMsg := 'there was an internal file system
error';
    128: GetErrMsg := 'printing aborted at user request';
  end;
end; ( function GetErrMsg )

procedure IOCheck(result:OSErr);
var
  ignore:integer;
  errorString:Str255;
begin
  if result <> NoErr then
    begin
      NumToString(result,errorString);
      ParamText('Macintosh OS Error #:', errorString, 'Macin-
tosh OS Error Desc:', GetErrMsg(result));
      ignore := StopAlert(StopDialogID,NIL);
    end
  end; ( procedure IOCheck )

procedure LaunchFailed(errNo:OSErr);
var
  ignore:integer;
  errorString:Str255;
begin
  NumToString(errNo,errorString);
  ParamText('Launch Error #:', errorString, '', '');
  ignore := StopAlert(StopDialogID,NIL);
end; ( procedure LaunchFailed )

procedure NotFoundMsg(str1,str2:Str255);
var
  ignore: integer;
begin
  ParamText('Please help locate', str1, str2, 'using the
following dialog...');
  ignore := StopAlert(StopDialogID,NIL);
end; ( procedure LaunchFailed )

( ***** Configuration routines ***** )
function ReadDefaultData: integer;
( Open the launch data file and read the data necessary for
locating launch application and associated document (if any).
)

```



```

var
  theErr: OSErr;
  myAppFile: AppFile;
  myRefNum: integer;
  theResult: integer;
  appMsg: integer;
  appCount: integer;
begin
  theResult := 0; ( be optimistic )
  CountAppFiles(appMsg, appCount);
  if appCount < 1 then
    begin
      theResult := 1;
      AlertMsg('Dispatcher must be launched', 'from a data
document');
      ( could create a new data document file instead )
    end
  else
    begin
      GetAppFiles(1, myAppFile);
      ( open the file for reading )
      myPB.ioCompletion := nil;
      myPB.ioNamePtr := @myAppFile.fName;
      myPB.ioVRefNum := myAppFile.vRefNum;
      myPB.ioPermssn := fsRdWrPerm; ( read/write permission )
      myPB.ioMisc := nil; ( use volume buffer )
      theErr := PBOpen(@myPB, false);
      if theErr <> noErr then
        begin
          IOCheck(theErr);
          theResult := 1;
        end
      else
        begin
          myRefNum := myPB.ioRefNum;
          ( read the application launch data )
          myPB.ioCompletion := nil;
          myPB.ioRefNum := myRefNum;
          myPB.ioBuffer := @applData;
          myPB.ioReqCount := sizeof(LaunchData);
          myPB.ioPosMode := fsAtMark;
          myPB.ioPosOffset := 0;
          theErr := PBRead(@myPB, false);
          if ((theErr <> noErr) or (myPB.ioActCount < 1))
            then
              begin
                applData.vName := 'Uninitialized';
                applData.dirID := 0;
                applData.fName := 'Uninitialized';
                applData.fType := 'APPL';
                applData.useIt := 1; ( not used )
              end;

            ( read the document launch data )
            myPB.ioCompletion := nil;
            myPB.ioRefNum := myRefNum;
            myPB.ioBuffer := @docLData;
            myPB.ioReqCount := sizeof(LaunchData);
            myPB.ioPosMode := fsAtMark;
            myPB.ioPosOffset := 0;
            theErr := PBRead(@myPB, false);
            if ((theErr <> noErr) or (myPB.ioActCount < 1))
              then
                begin
                  docLData.vName := 'Uninitialized';
                  docLData.dirID := 0;
                  docLData.fName := 'Uninitialized';
                  docLData.fType := 'TEXT';
                  docLData.useIt := 1;
                end;

            ( close the data file )
            myPB.ioCompletion := nil;
            myPB.ioRefNum := myRefNum;
            theErr := PBClose(@myPB, false);

```

```

      end; ( if data file opened OK )
    end; ( if app files count is > 0 )
    ReadDefaultData := theResult;
  end; ( procedure ReadDefaultData )

function WriteDefaultData: integer;
( Open the launch data file and write the data necessary for
locating launch application and associated document (if any).
This should take place any time the data is changed. )
var
  theErr: OSErr;
  myAppFile: AppFile;
  myRefNum: integer;
  theResult: integer;
begin
  theResult := 0; ( be optimistic )
  GetAppFiles(1, myAppFile);
  ( open the file for reading/writing )
  myPB.ioCompletion := nil;
  myPB.ioNamePtr := @myAppFile.fName;
  myPB.ioVRefNum := myAppFile.vRefNum;
  myPB.ioPermssn := fsRdWrPerm; ( read/write permission )
  myPB.ioMisc := nil; ( use volume buffer )
  theErr := PBOpen(@myPB, false);
  if theErr <> noErr then
    begin
      IOCheck(theErr);
      theResult := 1;
    end
  else
    begin
      myRefNum := myPB.ioRefNum;
      ( write the application launch data )
      myPB.ioCompletion := nil;
      myPB.ioRefNum := myRefNum;
      myPB.ioBuffer := @applData;
      myPB.ioReqCount := sizeof(LaunchData);
      myPB.ioPosMode := fsAtMark;
      myPB.ioPosOffset := 0;
      theErr := PBWrite(@myPB, false);
      if ((theErr <> noErr) or (myPB.ioActCount < 1)) then
        begin
          IOCheck(theErr);
          theResult := 1;
        end
      else
        begin
          ( write the document launch data )
          myPB.ioCompletion := nil;
          myPB.ioRefNum := myRefNum;
          myPB.ioBuffer := @docLData;
          myPB.ioReqCount := sizeof(LaunchData);
          myPB.ioPosMode := fsAtMark;
          myPB.ioPosOffset := 0;
          theErr := PBWrite(@myPB, false);
          if ((theErr <> noErr) or (myPB.ioActCount < 1))
            then
              begin
                IOCheck(theErr);
                theResult := 1;
              end;
            ( if bad document write )
          end;
          ( if application write OK )
          ( close the data file )
          myPB.ioCompletion := nil;
          myPB.ioRefNum := myRefNum;
          theErr := PBClose(@myPB, false);
        end;
        ( if data file opened OK )
        WriteDefaultData := theResult;
      end; ( procedure WriteDefaultData )

function FindApp1File: integer;
( Given the info in the data document, try to find the
target
application's working directory reference number. If it can't

```

```

be found, then prompt the user to find it. )
var
  theErr: OSErr;
  theResult: integer;
  reply: SFRReply;
  topLeft: Point;
  fileFilter: SFTypeList;
  appdone, appfound: integer;
  theName: Str255;
  theIndex: integer;
  theRefNum: integer;
begin
  theResult := 0; ( be optimistic )
  ( see if the volume is mounted )
  theIndex := 1;
  appdone := 0; appfound := 0; theName := '';
  repeat
    myHPB.ioVolIndex := theIndex;
    myHPB.ioCompletion := NIL;
    myHPB.ioNamePtr := @theName;
    theErr := PBHGetVInfo(@myHPB, false);
    if theErr <> noErr then
      appdone := 1; ( no more volumes to check )
    if theName = applData.vName then
      begin
        appfound := 1; appdone := 1;
        theRefNum := myHPB.ioVRefNum;
      end;
    theIndex := theIndex + 1;
  until appdone > 0;

  if appfound = 0 then
    theResult := 1
  else
    begin
      ( see if file can be found )
      myHPB.ioCompletion := NIL;
      myHPB.ioNamePtr := @applData.fName;
      myHPB.ioVRefNum := theRefNum; ( from call above )
      myHPB.ioDirIndex := 0; ( use file name, not index )
      myHPB.ioDirID := applData.dirID; ( use stored Dir ID )
      theErr := PBHGetFInfo(@myHPB, false);
      if (theErr <> noErr) then
        begin
          ( IOCheck(theErr); )
          theResult := 1;
        end
      else
        begin
          ( get the current application WD Ref Num )
          myWDPB.ioCompletion := nil;
          myWDPB.ioNamePtr := nil; ( directory name )
          myWDPB.ioVRefNum := theRefNum;
          myWDPB.ioWDProcID := longint('ERIK');
          myWDPB.ioWDDirID := applData.dirID;
          theErr := PBOpenWD(@myWDPB, false);
          applData.vRefNum := myWDPB.ioVRefNum;
          IOCheck(theErr);
        end; ( getting the current WD Ref Num )
      end; ( if volume was found OK )

      ( if application couldn't be found using default information, make the user find it manually. )
      if theResult <> 0 then
        begin
          NotFoundMsg('the program file: ', applData.fName);
          topLeft.h := 80; topLeft.v := 70;
          fileFilter[0] := applData.fType;
          SFGetFile(topLeft, '', NIL, 1, fileFilter, NIL, reply);
          if reply.Good then
            begin
              theResult := 0; ( reset the error flag )
              applData.vRefNum := reply.vRefNum; ( WD RefNum )
              applData.fName := reply.fName;
            end
          else
            begin
              theResult := 1;
            end
          end;
        end;
    end;
  end;
end;

```

```

applData.fType := reply.fType;
( look up new volume name )
myHPB.ioCompletion := NIL;
myHPB.ioNamePtr := @applData.vName;
myHPB.ioVRefNum := applData.vRefNum;
myHPB.ioVolIndex := 0; ( use vRefNum, not name )
theErr := PBHGetVInfo(@myHPB, false);
IOCheck(theErr);
( look up new application directory ID )
myInfoPB.ioCompletion := nil;
myInfoPB.ioNamePtr := @applData.fName;
myInfoPB.ioVRefNum := applData.vRefNum;
myInfoPB.ioDirIndex := 0;
myInfoPB.ioDirID := 0;
theErr := PBGetCatInfo(@myInfoPB, false);
IOCheck(theErr);
applData.dirID := myInfoPB.ioFLParID;
( update the default data )
if WriteDefaultData <> 0 then
  theResult := 1;
end ( if reply is good )
else
  begin
    theResult := 1;
  end; ( user cancelled application lookup )
end; ( if default application location not valid )
FindAppFile := theResult;
end; ( procedure FindAppFile )

function FindDocFile: integer;
( Given the info in the data document, try to find launch document's working directory reference number. If it can't be found, then prompt the user to find it. )
var
  theErr: OSErr;
  theResult: integer;
  reply: SFRReply;
  topLeft: Point;
  fileFilter: SFTypeList;
  docdone, docfound: integer;
  theName: Str255;
  theIndex: integer;
  theRefNum: integer;
begin
  theResult := 0; ( be optimistic )

  ( see if there is an associated document with application )
  if (docLData.useIt = 1) then ( launch with associated document )
    begin
      ( see if the volume is mounted )
      theIndex := 1;
      docdone := 0; docfound := 0; theName := '';
      repeat
        myHPB.ioVolIndex := theIndex;
        myHPB.ioCompletion := NIL;
        myHPB.ioNamePtr := @theName;
        theErr := PBHGetVInfo(@myHPB, false);
        if theErr <> noErr then
          docdone := 1; ( no more volumes to check )
        if theName = docLData.vName then
          begin
            docfound := 1; docdone := 1;
            theRefNum := myHPB.ioVRefNum;
          end;
        theIndex := theIndex + 1;
      until docdone > 0;

      if docfound = 0 then
        theResult := 1
      else
        begin
          ( see if the file can be found in its directory )
          myHPB.ioCompletion := NIL;

```

```

myHPB.ioNamePtr := @docLData.fName;
myHPB.ioVRefNum := theRefNum; ( from call above )
myHPB.ioFDirIndex := 0; (use file name, not index)
myHPB.ioDirID := docLData.dirID; (use stored Dir ID)
theErr := PBHGetFInfo(@myHPB,false);
if (theErr <> noErr) then
begin
  ( IOCheck(theErr); )
  theResult := 1;
end
else
begin
  ( get the current document WD Ref Num )
  myWDPB.ioCompletion := nil;
  myWDPB.ioNamePtr := nil; ( directory name )
  myWDPB.ioVRefNum := theRefNum;
  myWDPB.ioWDProcID := longint('ERIK');
  myWDPB.ioWDDirID := docLData.dirID;
  theErr := PBOpenWD(@myWDPB, false);
  docLData.vRefNum := myWDPB.ioVRefNum;
  IOCheck(theErr);
end; ( getting the current WD Ref Num )
end; ( if volume was found OK )

( if document couldn't be found using default information,
make the user find it manually. )
if theResult <> 0 then
begin
  NotFoundMsg('the associated document:
',docLData.fName);
  topLeft.h := 80; topLeft.v := 70;
  fileFilter[0] := docLData.fType;
  SFGetFile(topLeft,'',NIL,-1,fileFilter,NIL,reply);
  if reply.Good then
begin
  theResult := 0; ( reset the error flag )
  docLData.vRefNum := reply.vRefNum;
  docLData.fName := reply.fName;
  docLData.fType := reply.fType;
  ( look up new volume name )
  myHPB.ioCompletion := nil;
  docLData.vName := '';
  myHPB.ioNamePtr := @docLData.vName;
  myHPB.ioVRefNum := docLData.vRefNum;
  myHPB.ioVolIndex := 0; ( use vRefNum, not name )
  theErr := PBHGetVInfo(@myHPB,false);
  ( look up new document parent directory ID )
  myInfoPB.ioCompletion := nil;
  myInfoPB.ioNamePtr := @docLData.fName;
  myInfoPB.ioVRefNum := docLData.vRefNum;
  myInfoPB.ioFDirIndex := 0;
  myInfoPB.ioDirID := 0;
  theErr := PBGetCatInfo(@myInfoPB,false);
  IOCheck(theErr);
  docLData.dirID := myInfoPB.ioFLParID;
  ( parent directory ID )

  ( update the default data )
  if WriteDefaultData <> 0 then
    theResult := 1;
end ( if reply is good )
else
begin
  theResult := 1;
end; ( user cancelled document lookup )
end; ( if default document location not valid )
end; ( if associated document exists )
FindDocFile := theResult;
end; ( procedure FindDocFile )

function EditDefaults: integer;
var
  itemHit: integer;
  itemType: integer;

```

```

  itemRect: Rect;
  df1tdlg: DialogPtr;
  itemHdl1, itemHdl2: Handle; ( edittext handles )
  itemHdl3: Handle; ( check box dialog handle )
  cBoxHdl1: ControlHandle; ( check box control handle )
  theResult: integer;
begin
  theResult := 0;
  ( get dialog & edittext handles )
  df1tdlg := GetNewDialog(df1tdlgID, nil, Pointer(-1));
  GetDItem(df1tdlg, 4, itemType, itemHdl1, itemRect);
  GetDItem(df1tdlg, 5, itemType, itemHdl2, itemRect);
  GetDItem(df1tdlg, 6, itemType, itemHdl3, itemRect);
  cBoxHdl1 := ControlHandle(itemHdl3);
  SetIText(itemHdl1, applData.fName); ( app filename )
  SetIText(itemHdl2, docLData.fName); ( doc filename )
  SetCtlValue(cBoxHdl1, docLData.useIt); ( use/don't use
document )

  ( put up dialog )
  itemHit := 0;
  while ((itemHit > 3) or (itemHit < 1)) do
begin
  ModalDialog(nil, itemHit);
  if itemHit = 6 then ( toggle the check box )
begin
  if (GetCtlValue(cBoxHdl1) = 0) then
    SetCtlValue(cBoxHdl1, 1)
  else
    SetCtlValue(cBoxHdl1, 0);
end;
end; ( while )
  ( get dialog results )
  case itemHit of
    1: begin ( use names entered in dialog )
        ( application name )
        GetIText(itemHdl1, applData.fName);
        ( document name )
        GetIText(itemHdl2, docLData.fName);
      end;
    2: begin ( cause a search for new files )
        applData.fName := 'XXXXXXXX';
        docLData.fName := 'XXXXXXXX';
      end;
    3: theResult := 1; ( cancel the launch )
      end;
  if (theResult = 0) then
begin
    docLData.useIt := GetCtlValue(cBoxHdl1);
    theResult := WriteDefaultData; ( update the defaults )
  end;
  DisposDialog(df1tdlg);
  EditDefaults := theResult;
end; ( EditDefaults )

function TestCommandKey: integer;
( if command is held down while starting, the user will
be allowed to edit the default settings. The file data
has already been read at this point. If the user cancels,
the data defaults are not changed. )
var
  theResult: integer;
  eventReady: boolean;
  myEvent: EventRecord;
begin
  theResult := 0;
  eventReady := GetNextEvent(everyEvent, myEvent);
  if (BAND(myEvent.modifiers,cmdkey) > 0) then
    theResult := EditDefaults;
  TestCommandKey := theResult;
end; ( TestCommandKey )

( ***** Transfer and launch routines ***** )

```

function ResetFinderInfo: integer;
 (This is an example of how the finder parms could be changed
 without allocating a new structure in the system heap.
 However, the changes could not exceed the size of the existing
 heap object. This routine is not used.)

```
var
  fInfoHdl: Handle;
  hSize: integer;
  appParmRec: APRec;      ( my Finder info )
  theResult: integer;
begin
  theResult := 0; ( be optimistic )
  ( get AppParmHandle, lock it, and copy parms record )
  fInfoHdl := Handle(AppParmGVar);
  fInfoHdl := Handle(fInfoHdl);
  HLock(fInfoHdl);
  hSize := GetHandleSize(fInfoHdl);
  BlockMove(Pointer(fInfoHdl), Pointer(@appParmRec), hSize);
  if (appParmRec.count > 0) then
    begin
      if (length(appParmRec.files[1].fName) >=
        length(docLData.fName)) then
        begin
          ( now change the launch parameters )
          appParmRec.files[1].fName := docLData.fName;
          appParmRec.files[1].fType := docLData.fType;
          appParmRec.files[1].vRefNum := docLData.vRefNum;
          BlockMove(Pointer(@appParmRec), Pointer(fInfoHdl), hSize);
        end
      end
    else
      begin
        AlertMsg('Unable to reset', 'launch file parameters');
        theResult := 1;
      end;
  HUnlock(fInfoHdl);
  ResetFinderInfo := theResult;
end; ( ResetFinderInfo )
```

function ZeroFinderInfo: integer;
 (If no associated documents are needed, then the number of
 files in app parm handle block must be set to zero, and
 the file types set to zero so the Finder can clean up when
 it gets control back. The number of files will always be 1
 when this routine is called.)

```
var
  fInfoHdl: Handle; ( Finder info handle )
  hSize: integer;
  appParmRec: APRec; ( my Finder info )
  theResult: integer;
begin
  theResult := 0; ( be optimistic )
  ( get handle, lock it, and copy parms record )
  fInfoHdl := Handle(AppParmGVar);
  fInfoHdl := Handle(fInfoHdl);
  HLock(fInfoHdl);
  hSize := GetHandleSize(fInfoHdl);
  BlockMove(Pointer(fInfoHdl), Pointer(@appParmRec), hSize);
  if (appParmRec.count > 0) then
    begin
      ( now change the launch parameters )
      appParmRec.count := 0;
      appParmRec.files[1].fType := OSType(longint(0));
      ( write it back out )
      BlockMove(Pointer(@appParmRec), Pointer(fInfoHdl), hSize);
    end;
  HUnlock(fInfoHdl);
  ZeroFinderInfo := theResult;
end; ( ZeroFinderInfo )
```

function ReplaceFinderInfo: integer;
 (To prepare for the launch, the finder launch information
 must be reset to hold the name and type of the database
 startup file. This routine disposes the old AppParmHandle heap

object and allocates a new one on the system heap. Returns 0
 if successful, 1 if an error occurred.)

```
var
  fInfoHdl: Handle; ( handle to Finder info )
  sysZone: THz; ( system heap zone pointer )
  appZone: THz; ( application heap zone pointer )
  hSize: integer;
  appParmRec: APRec; ( my Finder info )
  theErr: OSErr;
  theResult: integer;
  saveAppParm: ^longint; ( pointer used to access $AEC )
begin
  theResult := 0; ( be optimistic )
  ( get handle, lock it, and copy parms record )
  fInfoHdl := Handle(appParmGVar);
  fInfoHdl := Handle(fInfoHdl);
  HLock(fInfoHdl);
  hSize := GetHandleSize(fInfoHdl);
  BlockMove(Pointer(fInfoHdl), Pointer(@appParmRec), hSize);

  ( dispose the old handle in the system zone )
  HUnlock(fInfoHdl);
  appZone := GetZone;
  sysZone := SystemZone;
  SetZone(sysZone);
  DisposHandle(fInfoHdl);

  ( allocate a new handle in the system zone )
  hSize := 13 + length(docLData.fName); ( enough for one doc )
  fInfoHdl := NewHandle(hSize);
  theErr := MemError;
  SetZone(appZone);
  if theErr <> noErr then
    begin
      IOCheck(theErr);
      theResult := 1;
    end
  else
    begin
      HLock(fInfoHdl);
      ( put the new handle back at $AEC )
      saveAppParm := Pointer(AppParmGVar);
      saveAppParm^ := longint(fInfoHdl);

      ( now change the launch parameters )
      appParmRec.message := 0; ( open it )
      appParmRec.count := 1; ( number of documents )
      appParmRec.files[1].vRefNum := docLData.vRefNum;
      appParmRec.files[1].fType := docLData.fType;
      appParmRec.files[1].version := false;
      appParmRec.files[1].unused := false;
      appParmRec.files[1].fName := docLData.fName;
      BlockMove(Pointer(@appParmRec), Pointer(fInfoHdl),
        hSize);
      HUnlock(fInfoHdl);
    end; ( if sys heap handle allocated OK )
  ReplaceFinderInfo := theResult;
end; ( ReplaceFinderInfo )
```

function LaunchIt(pLnch: pLaunchStruct): OSErr;
 (from tech note #52 & 126)
 INLINE \$205F, \$A9F2, \$3E80;

procedure DoLaunch;
 (Launch the application, given the information in the
 launch data records for the application & document.)

```
var
  fileInfo: FInfo;
  ignore: integer;
  myLaunch: LaunchStruct;
  theErr: OSErr;
begin
  ( set default working directory to application's folder )
  myWDPB.ioCompletion := NIL;
```

```

myWDPB.ioNamePtr := NIL;
myWDPB.ioVRefNum := applData.vRefNum;
theErr := PBSetVol(@myWDPB,false);

( get finder flags for launch record )
myInfoPB.ioNamePtr := @applData.fName;
myInfoPB.ioVRefNum := applData.vRefNum;
myInfoPB.ioFDirIndex := 0; ( use name instead of index )
myInfoPB.ioDirID := 0;
theErr := PBGetCatInfo(@myInfoPB,false);

( now launch the application )
myLaunch.pfName := @applData.fName;
myLaunch.param := 0;
myLaunch.LC := 'LC';
myLaunch.extBlockLen := 6;
myLaunch.LaunchFlags := $00000000; ($C0000000 for sublaunch)
myLaunch.fFlags := myInfoPB.ioFInfo.fFlags; ( from
GetCatInfo )

theErr := LaunchIt(@myLaunch);
if theErr < 0 then
    LaunchFailed(theErr);
end; ( procedure DoLaunch )

function PrepareForLaunch: integer;
( set up the AppParmsHandle data )
var
    theResult: integer;
begin
    theResult := 0;
    if (docLData.useIt = 1) then ( application & document )
        theResult := ReplaceFinderInfo
    else ( application only )
        theResult := ZeroFinderInfo;
    PrepareForLaunch := theResult;
end; ( function PrepareForLaunch )

procedure Initialize;
( initialize managers & globals )
begin
    ( initialize the managers )
    InitGraf(@thePort);
    InitFonts;
    InitWindows;
    InitMenus;
    TEInit;
    InitDialogs(NIL);
    FlushEvents(everyEvent,0);
end; ( Initialize )

( main program )
begin
    Initialize;
    lcnt := 1; done := 0;
    while done = 0 do
        begin
            case lcnt of
                1: done := ReadDefaultData;
                2: done := TestCommandKey;
                3: done := FindApplFile;
                4: done := FindDocFile;
                5: done := PrepareForLaunch;
                6: DoLaunch; ( returns only if sublaunched )
            end;
            lcnt := lcnt + 1;
            if lcnt > 6 then done := 1;
        end; ( while )
    end. ( dispatcher )

Listing: dispatcher.r

/*dispatcher.r
Copyright Verity Software Systems 1989

```

```

All rights reserved.
Resource definitions for dispatcher application
Rez format

*/
#include "Types.r"
resource 'BNDL' (128) (
    'LCHA',
    0,
    ( /* array TypeArray: 2 elements */
        /* [1] */
        'ICN#',
        ( /* array IDArray: 2 elements */
            /* [1] */
            0, 128,
            /* [2] */
            1, 129
        ),
        /* [2] */
        'FREF',
        ( /* array IDArray: 2 elements */
            /* [1] */
            0, 130,
            /* [2] */
            1, 131
        )
    )
);
resource 'ICN#' (128, "Dispatcher Application Icon") (
    ( /* array: 2 elements */
        /* [1] */
        $"FFFF FFFE 8000 0003 8120 0003 8120 0003"
        $"8220 0003 8440 0003 8880 07C3 8118 1823"
        $"861C 2023 880F FF23 8006 0023 8005 0043"
        $"8004 8043 8004 4083 800C 2083 800C 0103"
        $"8014 0203 8014 0603 8024 0A03 8024 1203"
        $"8020 7203 8021 D203 801E 1203 8000 1203"
        $"8000 1203 8000 1203 8000 3303 8000 2103"
        $"8000 2103 8000 2103 FFFF FFFF 7FFF FFFF",
        /* [2] */
        $"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
        $"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
        $"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
        $"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
        $"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
        $"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
        $"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
    )
);
resource 'ICN#' (129, "Dispatcher Document Icon") (
    ( /* array: 2 elements */
        /* [1] */
        $"FFFF FFFE 8000 0003 8000 1E03 8000 0183"
        $"8000 1C43 8000 0243 83C0 0123 8430 18A3"
        $"8408 38A3 84FF F0A3 8400 6003 8200 A003"
        $"8201 2003 8102 2003 8104 3003 8080 3003"
        $"8040 2803 8060 2803 8050 2403 8048 2403"
        $"804E 0403 8048 8403 8048 7803 8048 0003"
        $"8048 0003 8048 0003 80CC 0003 8084 0003"
        $"8084 0003 8084 0003 FFFF FFFF 7FFF FFFF",
        /* [2] */
        $"FFFF FFFE FFFF FFFF FFFF FFFF FFFF FFFF"
        $"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
        $"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
        $"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
        $"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
        $"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
        $"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
    )
);
resource 'FREF' (130) (
    'APPL',
    0,

```

```

"Dispatcher Application"
);
resource 'FREF' (131) (
    'LCHD',
    1,
    "Dispatcher Document"
);
data 'LCHA' (0) (
    "$4469 7370 6174 6368 6572 2041 7070 6C69"
    /* Dispatcher Application */
    "$6361 7469 6F6E 202D 2043 6F70 7972 6967"
    /* Copyright 1989 Verity Software Systems. */
    "$6874 2031 3938 3920 5665 7269 7479 2053"
    "$6F66 7477 6172 6520 5379 7374 656D 732E"
    "$2052 6F67 6572 2041 2E20 486F 7274 6F6E"
    /* Roger A. Horton */
    "$202D 2041 6C6C 2072 6967 6874 7320 5265"
    /* - All rights Reserved */
    "$7365 7276 6564"
);
resource 'DLOG' (200, "Configuration Settings") (
    (38, 116, 312, 390),
    altDBoxProc,
    visible,
    noGoAway,
    0x0,
    200,
    "Configuration Settings"
);
resource 'DITL' (200, "Configuration Settings") (
    /* array DITLarray: 13 elements */
    /* [1] */
    (184, 24, 204, 84),
    Button (
        enabled,
        "OK"
    ),
    /* [2] */
    (184, 109, 204, 169),
    Button (
        enabled,
        "Search"
    ),
    /* [3] */
    (184, 192, 204, 252),
    Button (
        enabled,
        "Cancel"
    ),
    /* [4] */
    (74, 31, 92, 239),
    EditText (
        enabled,
        "4th Dimension® v1.0.6"
    ),
    /* [5] */
    (128, 30, 146, 238),
    EditText (
        enabled,
        "FPR"
    ),
    /* [6] */
    (151, 56, 171, 212),
    CheckBox (
        enabled,
        "Open App Document"
    ),
    /* [7] */
    (4, 81, 24, 182),
    StaticText (
        enabled,
        "o Dispatcher o"
    ),
    /* [8] */

```

```

(25, 75, 45, 190),
    StaticText (
        enabled,
        "Current Settings"
    ),
    /* [9] */
    (51, 31, 70, 239),
    StaticText (
        enabled,
        "Application Name:"
    ),
    /* [10] */
    (106, 32, 125, 239),
    StaticText (
        enabled,
        "Document Name:"
    ),
    /* [11] */
    (220, 30, 238, 249),
    StaticText (
        enabled,
        "® Verity Software Systems 1989"
    ),
    /* [12] */
    (239, 15, 256, 128),
    StaticText (
        enabled,
        "by Roger Horton"
    ),
    /* [13] */
    (239, 132, 256, 263),
    StaticText (
        enabled,
        "All Rights Reserved"
    )
);
resource 'ALRT' (201) (
    (80, 100, 210, 400),
    201,
    /* array: 4 elements */
    /* [1] */
    OK, visible, sound1,
    /* [2] */
    OK, visible, sound1,
    /* [3] */
    OK, visible, sound1,
    /* [4] */
    OK, visible, sound1
);
resource 'DITL' (201) (
    /* array DITLarray: 2 elements */
    /* [1] */
    (100, 125, 120, 175),
    Button (
        enabled,
        "OK"
    ),
    /* [2] */
    (10, 70, 90, 290),
    StaticText (
        disabled,
        "~0 \n^1 \n^2 \n^3"
    )
);
resource 'SIZE' (-1) (
    dontSaveScreen,
    ignoreSuspendResumeEvents,
    dontDoOwnActivate,
    128000,
    128000
);

```



Pascal Procedures

AACK gives AppleTalk Shell



Ajay Nath
Oakland Gardens, NY
Volume 5, Number 12

[Ajay Nath has two B.A.s, one in Chemistry and one in Mathematics. He is currently a second year medical student and is using Macs to develop Color Vision tests.]

AACK OR ATApp?

Example AppleTalk programs have appeared before in MacTutor but none recently except in one of the HyperCard columns. The introduction of MultiFinder, a new interface to the AppleTalk routines and new features added to AppleTalk itself, have changed the way applications should use AppleTalk. In this article I present a program called AACK, the "AppleTalk Application Construction Kit", which was written in LightSpeed Pascal 2.0, which demonstrates how to use the new AppleTalk interface and how to use the background features of MultiFinder. The resources of the program were created by ResEdit and stored in a file called "aack.rsrc", a rez description of the resources is found in the file "resources.r". The code of the program is in 4 files/units as follows:

Options	File (by build order)	Size
	µRuntime.Lib	5870
	Interface.lib	8104
	ABPackage.lib	4692
	nAppleTalk.lib	670
D N V R	AppleTalk.p	0
D N V R	globals.p	0
D N V R	lowlevelatprocs.p	234
D N V R	hooks.p	124
D N V R	main.p	3752

FILE/UNIT	CONTENTS
globals.p	global declarations
lowlevelATprocs.p	low level procs
hooks.p	code specific to the application
main.p	generic application code

The code is designed to be easily modifiable. By simply changing code in the file hooks.p the nature of the applications produced can be changed. For example, in the file main.p, the code that handles menu selections look like:

```
IF HookedMenuChoice(theMenu, theItem) = FALSE THEN BEGIN
CASE theMenu OF
  APPLMENUID:
    DoAppleMenu(theItem);
  FILEMENUID:
    DoFileMenu(theItem);
```

```
SPECIALMENUID:
  DoSpecialMenu(theItem);
  OTHERWISE
END;
END;
```

where the function HookedMenuChoice is declared in the file/unit 'hooks.p'. It returns false if it DIDN'T handle the menu choice, it allows the programmer to 'hook' or handle menu choices before the applications generic code does and to override the generic codes response. For example, the generic code of AACK simply puts up a windows, some menus, and allows the user to see others of his type on the network and to confirm their presence. There is a menu titled "Special" which has items labeled "LookUp", "Confirm" and "Send". The generic code handles the first two items (this can be overridden) but does nothing but beep when "Send" is chosen. By changing the code in the hooks.p file, you can create an application that does something else with AppleTalk, perhaps sending strings through the net as was shown in a previous MacTutor article. There is obviously a code/speed overhead introduced by using this method, but its far more easy to get something to work and to later go back and optimize than to start each application from scratch. (Maybe I should have called it "ATApp: The Expandable Macintosh AppleTalk Application"?)

Bypassing Generic (Brand?) Code OR Over-Rid'em Cowboy!

Most of the generic code is trivial stuff to MacTutor readers and I have left out some things such as handling DAs which is left to the reader as an exercise. The code does use LSPs ability to have compiler variables. I declare a compiler option called "TALK_DEBUG" and set it to TRUE when I want the compiler to include certain debugging code in the app. Basically, the generic code does a "SetUp" routine and lets the code from the "Hooks" unit do any "SetUp" - i.e. - the generic routine SetUpGlobals does its "SetUps" and then calls the procedure "SetUpHooksGlobals", which lets the Hooks code do its "SetUps". Similarly the generic code lets the Hooks code do any Window and AppleTalk "SetUps" after it has done its own such "SetUps". The generic code also checks to see if the Hooks code has decided to take over the duties of taking care of the applications window by examining a global Boolean variable called "UserWindowProcsChanged" which the Hooks code can set to true in its "SetUp" code. For example when activating the UserWindow the generic code does the following:

```
IF UserWindowProcsChanged THEN
  HooksActivateUserWindowProc (Let hook code do its magic)
ELSE
  LActivate(T, NameListHd1);
```

One consequence of this is that even if the Hooks code wants to override only one of the procedures that controls the windows activation, deactivation, updating, or disposal, it must override them ALL. You only have to copy and paste the generic code from the file main.p into the file hooks.p to mimic the generic codes handling of the other procedures though. For example, you might write an app using AACK which receives/sends messages and overrides the way the generic code draws the windows contents to draw any received messages at the bottom of the window, but it wouldn't need to override any other of the generic codes window handling routines. You could just copy the other generic code routines from the file main.p and pasted them into the right places in the file hooks.p.

Generic AppleTalk Code OR AppleTalk Is cheap

The AppleTalk generic code does the following:

- 1) calls MPPOpen and then SysEnviron which returns to us among other information, the type of machine we're running on and the version of AppleTalk available. We exit if we're not running on at least a 512KE, and if we are, set a boolean variable called "NewCallsExist" to true if the AppleTalk driver version supports the new AppleTalk calls
- 2) opens an ATP socket
- 3) calls WaitNextEvent a few times to make sure our dialog that lets the user select a name comes up in front and lets the user register on the network with a name (1->31 chars)
- 4) depending on the setting of a compiler variable called "TALK_DEBUG" tries to call PSetSelfSend, a new AppleTalk call which lets you send AppleTalk packets to your own node. This is useful if your network consists of one machine as mine does, unfortunately this feature is implemented only on machines with the drivers with version numbers >= 48, (actually its listed as 48 in IM V but 45 in the MPW Assembler Equates, which is it Apple?)
- 5) then the generic code lets the Hooks unit do any AppleTalk "SetUps", (such as post asynchronous GetRequests if it was going to receive requests)

If all of the above goes well then the generic code sets up its Menus, lets the Hooks unit do any of its menu setups, then the generic code sets up its window and lets the Hooks unit modify that window or set up any of its own and we're off into our event loop!

MultiFinder and AppleTalk OR ASYNC or swim!

The generic codes event loop calls WaitNextEvent (you already know how to check for the presence of WNE so I don't do this) and calls a procedure called "AppleTalkCallChecks" before it handles any events. This procedure is going to check for any ASYNC calls that may have finished and handle them. For example, the generic code calls PLookupName ASYNChronously, this allows the user to start a Lookup to see if any other users of the same type are on the network, and work on other things while the Lookup is begin done. This IS after all is what MultiFinder is for! The tricky part of making ASYNC calls is knowing when they've finished and taking care of them when they do finish. I solve this problem by using Completion

Routines and special data structures. I got the idea from Tech Note #180 - "MultiFinder Miscellanea", in which it is suggested that applications making ASYNC calls store their A5 somewhere RELATIVE to the parameter block which they use to make the ASYNC calls, so that when their Completion Routines are executed they can load the applications A5 and set a global variable to indicate that an ASYNC call has finished. I use a similar approach. I define the following structure in the file "globals.p":

```
NameLookupRec = RECORD
  PbInUse: Boolean;
  CallDone: LongInt;
  xMPPb: MPPParamBlock;
END;
```

When a call is made using a record of this type the field PbInUse is set to TRUE and CallDone is set to zero. The Completion Routine that is executed when the call finishes sets the CallDone field to -1.

The routine "AppleTalkCallChecks" (called once through each event loop) just checks the CallDone field of the record and if it is non-zero we know that an ASYNC call has just completed. The advantages of this method are as follows:

- 1) the Completion Routine used is VERY simple, as it must be since Completion Routines cannot make any calls that affect memory, all it does is set the CallDone field to a non-zero value. The completion routine is declared in LSP in the following way in the file "lowlevelATprocs.p":

```
PROCEDURE InlineRoutine;
  INLINE
  $213C, $FFFF, $FFFF; ( MOVEA.L #-1, -(A0) set CallDone
  field)

PROCEDURE XCompletionRoutine;
  (When routine is called A0 = ptr to ParamBlock)
  BEGIN
    InlineRoutine;
  END;
```

- 2) we dont have to worry about whether our A5 is valid when our Completion Routine runs

The drawbacks of using this method are that we must check our ASYNC calls EVERY time through our event loop and that the parameter blocks we use to make them MUST be global variables so that we can check them.

Using the PREFERRED Interface OR Learning to AppleTalk all over again

We'd like to use the (new) 'preferred' AppleTalk interface because its more robust, and if we're running with the right versions of the AppleTalk drivers we can use the new AppleTalk calls. Information about the new AppleTalk calls is in IM V (and I suppose, in Inside AppleTalk which is available from APDA which I dont have). There are two different types of parameter blocks defined in the new interface, the ATPParamBlock and the MPPParamBlock, and you just fill in the fields necessary for the call you want to make and then pass a ptr to the ParamBlock to the function you're calling - i.e. -


```
err = PCallName (@ParamBlock, true or false for ASYNC);
```

The catch is that when filling out the fields in parameter blocks for OLD AppleTalk routines that you call with the NEW interface, you use the information from "Calling the AppleTalk Manager routines from Assembly Language" from IM II. Look at the pages in IM V which describe the new types, it shows which fields have to be filled with which values to make a call. Some field names have changed though, so be careful, make sure you've filled them out right - then check them again!

New (and Improved?) AppleTalk Calls OR Don't LookUp before you leave

A few new AppleTalk calls are quite useful to us, for example, the generic code of AACK does ASYNC LookUps, but what happens if the user quits BEFORE the LookUp completes? If the application just quit, then sooner or later that ASYNC call would complete and the Completion Routine (which is no longer around) would be called and BOOOOM! If we don't have access to the new calls then we just have to wait till the ASYNC call completes which we can do by:

```
while CallDone = 0 do
;
```

, which just waits till a particular call is done before proceeding. There is however a new call, PKillNBP, which allows us to kill pending calls, (but if it fails we will have to wait the call out). Note that IM V in some places incorrectly lists which ParamBlock, ATP or MPP to use, this call requires an MPPParamBlock. The generic code also uses the call PSetSelfSend to allow the user to receive/send packets to his own node, a feature of the new AppleTalk drivers which is useful to programmers, but we must be careful to restore the state of this flag - i.e. - if it wasn't enabled when we started, then we should disable it when we leave.

The future OR Everybody AppleTalks but nobody DOES anything with it

Should the PConfirmName calls be made ASYNC? I'll leave it to you as an exercise. I've already written an application (modifying ONLY the file hooks.p) which uses the AACK shell to receive/send strings (Str255s actually). If anyone is interested in seeing it, let Dave Smith at MacTutor know and I'll write it up, its code will be shorter than this article but it may take longer to explain.

Using PSetSelfSend on a Mac Plus OR AppleTalking to yourself on a Plus

I've found a way to get the new AppleTalk drivers running on my Mac Plus and I can test my AppleTalk code on my 'network of one' using PSetSelfSend to enable me to send packets to myself. Those of you who are thinking, "That's easy, just copy the DRVR resources out of the latest System File and paste them into the System File you use on your Mac Plus!", are thinking along the wrong lines. The new drivers are in the Mac

II ROM and perhaps the SE ROM (I don't have one to test). I'm not sure its a good idea to use the new drivers on a Mac Plus, but I haven't had any problems with my system (yet). Apple must have some reason not to make the new drivers available for the Mac Plus so I won't pass this information along unless anyone out there is interested and Apple says its OK (you've figured out how to do it though, right?).

I'd like to thank K.S. without whose love and support this article would not be possible.

Listing: globals.p

```
unit Globals;

interface

uses
    AppleTalk;

type
    (AppleTalk Types)
        NameLookUpRec = record
            PbInUse: Boolean;
            CallDone: LongInt;
            xMPPb: MPPParamBlock;
        end;

    (String Types)
        Str32Hdl = ^Str32Ptr;
        Str32Ptr = ^Str32;

const
    F = false;
    T = true;

(Constants used ONLY when debugging)
($IFC TALK_DEBUG )
    SENDSELF = 1;
{$ENDC}

(AppleTalk Constants)
    ANYZONE = '*';
    ANYOBJ = '=';
    ASYNC = T;
    DOVERIFY = 1;
    LOOKUPBUFFERSIZE = 1023;
    MAXNAMELENGTH = 31;
    MAXTOLOOKUP = 100;
    SYNC = F;
    XNCVERSION = 48;

(Dialog Constants)
    rABOUTLOGID = 128;

    rUSERNAMELOGID = 129;
    rUSERNAMEITEM = 4;

(Event Constants)
    MFEVENT = app4Evt;
    RESUMEMASK = 1;
    SUSPENDRESUMMSG = 1;

(Menu Constants)
    APPLMENUID = 128;
    ABOUTITEM = 1;
    FILEMENUID = 129;
    QUITITEM = 1;
    EDITMENUID = 130;
    SPECIALMENUID = 131;
    LOOKUPITEM = 1;
    CONFIRMITEM = 2;
    SENDITEM = 3;

(String Resource Constants)
```

```

rAPPUSERNAMESTRID = 128;
rCHOOUSERUSERSTRID = -16096;

var
  DoneFlag: Boolean;
  DragRect: Rect;
  Evt: EventRecord;

(Globals used ONLY when debugging)
($IFC TALK_DEBUG )
  DebugOnRect: Rect;
  OldSelfFlag: Byte;
  SelfSendOn: Boolean;
  SelfSendOnRect: Rect;
($ENDC)

(AppleTalk Globals)
  ConfirmString: Str255;
  ConfirmStringPos: Point;
  ConfirmStringRect: Rect;
  LookUpBuffer: array[0..LOOKUPBUFFERSIZE] of Integer;
  LookUpNamePb: NameLookUpRec;
  LookUpNTT: EntityName;
  LookUpString: Str255;
  LookUpStringPos: Point;
  LookUpStringRect: Rect;
  NBPsNTE: NamesTableEntry;
  NewCallsExist: Boolean;
  UserNTT: EntityName;
  UserSkt: Byte;

(List Globals)
  ListFrameRect, ListViewRect: Rect;
  MaxRows, NameListLength: Integer;
  NameListHdl: ListHandle;

(Event Globals)
  SleepTime: LongInt;

(Menu Globals)
  AppleMenu, FileMenu, EditMenu, SpecialMenu: MenuHandle;

(Window Globals)
  TextHeight: Integer;
  UserWindowFontInfo: FontInfo;
  UserWindowProcsChanged: Boolean;
  UserWindowRect: Rect;
  UserWindow, WhichWindow: WindowPtr;
  UserWRec: WindowRecord;

implementation
end.

Listing: lowlevelatprocs.p

unit LowLevelATProcs;

interface

  uses
    AppleTalk, Globals;

  procedure XCompletionRoutine;
  function NTTEExists (ntt2Confirm: EntityName; var theAddress:
    AddrBlock): OSErr;

  procedure DrawConfirmString;
  procedure DrawLookUpString;

implementation
  procedure InlineRoutine;
  inline
    $213C, $FFFF, $FFFF; ( MOVEA.L #-1, -(A0) set CallDone
    field)

```

```

  procedure XCompletionRoutine;
  (When routine is called A0 = ptr to ParamBlock)
  begin
    InlineRoutine;
  end;

  function NTTEExists (ntt2Confirm: EntityName; var theAddress:
    AddrBlock): OSErr;
  (This function test whether or not an NTT exists at a
  certain address and)
  ( returns its new address if it 'moved')
  var
    err: OSErr;
    localMPPPB: MPPParamBlock;
  begin
    with localMPPPB do
      begin
        ioCompletion := nil;
      ($IFC TALK_DEBUG )
        interval := 6;
        count := 2;
      ($ELSC)
        interval := 10;
        count := 10;
      ($ENDC)
        entityPtr := @ntt2Confirm; (NTT to confirm)
        confirmAddr := theAddress; (address to confirm)

        err := PConfirmName(@localMPPPB, SYNC);
        NTTEExists := err;
        if err = nbpConfDiff then
          (If the NTT 'moved' then return its new address)
          theAddress.aSocket := newSocket;
        end;
      end;
    end;

  procedure DrawConfirmString;
  (This proc draws the result of the last ConfirmName call)
  (It assumes that the current port is the user window)
  begin
    with ConfirmStringPos do
      begin
        EraseRect(ConfirmStringRect);
        MoveTo(h, v);
        DrawString('Confirm Status = ');
        DrawString(ConfirmString);
      end;
    end;
  end;

  procedure DrawLookUpString;
  (This proc draws the result of the last LookUpName call)
  (It assumes that the current port is the user window)
  begin
    with LookUpStringPos do
      begin
        EraseRect(LookUpStringRect);
        MoveTo(h, v);
        DrawString('LookUp Status = ');
        DrawString(LookUpString);
      end;
    end;
  end;
end.

```

Listing: hooks.p

```

unit Hooks;

interface

  uses
    AppleTalk, Globals, LowLevelATProcs;

```

```

const
  USERTYPE = 'NATH';

procedure SetUpHooksGlobals;
procedure SetUpHooksMenus;

function HookedMenuChoice (menu, item: Integer): Boolean;

procedure HooksActivateUserWindowProc;
procedure HooksCloseUserWindowProc;
procedure HooksDeactivateUserWindowProc;
procedure HooksUpdateUserWindowProc;
procedure HooksUserWindowDoContentHit;

procedure ActivateHooksWindow (theWindow: WindowPtr);
procedure CloseHooksWindows;
procedure DeactivateHooksWindow (theWindow: WindowPtr);
procedure DoHooksWindowContentHit;
function SetUpHooksWindows: Boolean;
procedure UpdateHooksWindow (theWindow: WindowPtr);

procedure HooksAppleTalkCallChecks;
function HooksAppleTalkGlobalsSetUp: Boolean;
procedure HooksCloseUpAppleTalk;
function HooksRegistered: Boolean;

implementation

procedure SetUpHooksGlobals;
  (This proc allows the hook code to init its globals)
begin
end;

procedure SetUpHooksMenus;
  (This proc allows the hook code to init its menus)
begin
end;

function HookedMenuChoice (menu, item: Integer): Boolean;
  (This proc allows the hook code to override menu choices
  which the generic code)
  (would normally handle, it returns T if it overrode a menu
  choice)
begin
  (Assume we're not handling the menu choice)
  HookedMenuChoice := F;
end;

procedure HooksActivateUserWindowProc;
  (This proc is used to override the generic codes activation
  of the UserWindow)
begin
end;

procedure HooksCloseUserWindowProc;
  (This proc is used to override the generic codes closing of
  the UserWindow)
begin
end;

procedure HooksDeactivateUserWindowProc;
  (This proc is used to override the generic codes deactiva-
  tion of the UserWindow)
begin
end;

procedure HooksUpdateUserWindowProc;
  (This proc is used to override the generic codes updating of
  the UserWindow)
begin
end;

procedure HooksUserWindowDoContentHit;

```

```

  (This proc is used to override the generic codes handling of
  mouse hits)
  (in the UserWindow)
begin
end;

procedure ActivateHooksWindow (theWindow: WindowPtr);
  (This proc is used to activate any windows the hook code
  created)
begin
end;

procedure CloseHooksWindows;
  (proc is used to close any windows the hook code created)
begin
end;

procedure DeactivateHooksWindow (theWindow: WindowPtr);
  (proc is used to deactivate any windows hook code created)
begin
end;

procedure DoHooksWindowContentHit;
  (This proc is used to handle mouse hits in any windows the
  hook code created)
begin
end;

function SetUpHooksWindows: Boolean;
  (proc is used to setup windows hook code wants to create)
  (and it allows the hook code to set the UserWin-
  dowProcsChanged global)
  (it returns T if it succeeds)
begin
  (Assume error)
  SetUpHooksWindows := F;

  (Set to false to show that we dont change any of the
  UserWindow procs)
  UserWindowProcsChanged := F;

  (Return T since we succeeded)
  SetUpHooksWindows := T;
end;

procedure UpdateHooksWindow (theWindow: WindowPtr);
  (proc is used to update any windows the hook code created)
begin
end;

procedure HooksAppleTalkCallChecks;
  (proc is used to check any ASYNC AppleTalk calls hook code
  made)
begin
end;

function HooksAppleTalkGlobalsSetUp: Boolean;
  (This proc does any AppleTalk setups hook needs to make, it)
  (returns T if it succeeds)
begin
  (Assume error)
  HooksAppleTalkGlobalsSetUp := F;

  (Return T since we succeeded)
  HooksAppleTalkGlobalsSetUp := T;
end;

procedure HooksCloseUpAppleTalk;
  (proc does any AppleTalk closeups the hook needs to make)
begin
end;

function HooksRegistered: Boolean;
  (This allows the hook code to do any AppleTalk calls its

```

```

needs to make)
(after the User had registered. It returns T if it
succeed)
begin
  (Assume error)
  HooksRegistered := F;
  (Return T since we succeeded)
  HooksRegistered := T;
end;
end.

```

Listing: main.p

```

program AACK;

uses
  AppleTalk, Globals, LowLevelATProcs, Hooks;

procedure Crash;
(This proc is called if we crash)
begin
  ExitToShell;
end;

procedure SetUpToolbox;
(This proc inits the Mac toolbox)
begin
  MaxApplZone;
  MoreMasters;
  MoreMasters;
  MoreMasters;
  MoreMasters;
  InitGraf(@thePort);
  InitFonts;
  InitWindows;
  FlushEvents(everyEvent, 0);
  InitMenus;
  TEInit;
  InitDialogs(@Crash);
  InitCursor;
end;

procedure SetUpGlobals;
(This proc does any app global inits)
begin
  DoneFlag := F;

  DragRect := screenBits.bounds;
  with DragRect do
    begin
      top := top + 20; (NOTE! we really should use the
mbarheight)
      left := left + 4;
      bottom := bottom - 4;
      right := right - 4;
    end;

  (Assume the new calls dont exist)
  NewCallsExist := F;

  SleepTime := 10;

($IFC TALK_DEBUG )
  with DebugOnRect do
    begin
      top := 0;
      left := 0;
      bottom := 4;
      right := 4;
    end;

  with SelfSendOnRect do
    begin
      top := 5;

```

```

      left := 0;
      bottom := 9;
      right := 4;
    end;

($ENDC)
(Allow setup of any globals from the Hooks unit)
  SetUpHooksGlobals;
end;

procedure SetUpMenus;
(This proc sets up the generic codes menus)
begin
  AppleMenu := GetMenu(APPLEMENUID);
  AddResMenu(AppleMenu, 'DRVr');
  InsertMenu(AppleMenu, 0);

  FileMenu := GetMenu(FILEMENUID);
  InsertMenu(FileMenu, 0);

  EditMenu := GetMenu(EDITMENUID);
  InsertMenu(EditMenu, 0);

  SpecialMenu := GetMenu(SPECIALMENUID);
  InsertMenu(SpecialMenu, 0);

(Allow setup of any menus from the Hooks unit)
  SetUpHooksMenus;

  DrawMenuBar;
end;

procedure CloseWindows;
(This proc closes the apps windows)
begin
(Allow closing of any windows from the Hooks unit)
  CloseHooksWindows;

  if UserWindowProcsChanged then
    HooksCloseUserWindowProc (Let the hook code do it magic)
  else
    begin
      LDispose(NameListHdl);
      CloseWindow(UserWindow);
    end;
end;

function GotWindows: Boolean;
(This code sets up the apps windows, it returns T)
(if it succeeds)
var
  theCell: Cell;
  dBounds: Rect;
begin
  (Assume we fail)
  GotWindows := F;

  UserWindowRect := DragRect;
  with UserWindowRect do
    begin
      top := top + 20;
      left := left + 4;
      bottom := top + 200;
      right := left + 225;
    end;

  (Get the UserWindow)
  UserWindow := NewWindow(@UserWRec, UserWindowRect,
UserNTT.objStr, F, noGrowDocProc, WindowPtr(-1), T, 0);
  if UserWindow <> nil then
    begin
      SetPort(UserWindow);
      ClipRect(UserWindow^.portRect);
      TextFont(monaco);

```

```

    TextSize(9);

    GetFontInfo(UserWindowFontInfo);
    with UserWindowFontInfo do
        TextHeight := ascent + descent + leading;

    (Set the # of rows to display in our list)
    MaxRows := 10;

    (Set the current list length to 0)
    NameListLength := 0;

    with dBounds do
    begin
        top := 0;
        left := 0;
        bottom := NameListLength; (no rows for now)
        right := 1; (one column)
    end;

    theCell.h := 0;
    theCell.v := 0;

    (Set up the lists view rect)
    with ListViewRect do
    begin
        top := 10;
        left := 10;
        bottom := (TextHeight * MaxRows) +
        ListViewRect.top;
        right := (32 * CharWidth('w')) + 8;
    end;

    (Try to get a ListHandle)
    NameListHdl := LNew(ListViewRect, dBounds, theCell,
    0, UserWindow, T, F, F, T);

    if NameListHdl = nil then
    (We failed to get a ListHandle so close UserWindow and
    exit)
        CloseWindow(UserWindow)
    else
    begin
    (We got the ListHandle, allow only one selection at a time
    from it)
        NameListHdl^.selFlags := LOnlyOne;

    (Set up a rect to put a frame around our list)
        ListFrameRect := ListViewRect;
        with ListFrameRect do
        begin
            top := top - 1;
            left := left - 1;
            bottom := bottom + 1;
            right := right + 16;

    (Set up the pos to draw the LookUpName results at)
            with LookUpStringPos do
            begin
                h := left;
                v := bottom + TextHeight;
            end;

    (Set up the pos to draw the ConfirmName results at)
            with ConfirmStringPos do
            begin
                h := left;
                v := LookUpStringPos.v + TextHeight;
            end;

        end;

    (Set up the rect to erase the old LookUpName result)
        with LookUpStringRect do

```

```

        begin
            top := ListFrameRect.bottom + 1;
            left := UserWindow^.portRect.left;
            bottom := LookUpStringPos.v +
            UserWindowFontInfo.descent;
            right := UserWindow^.portRect.right;
        end;

    (Set up the rect to erase the old ConfirmName result)
        with ConfirmStringRect do
        begin
            top := LookUpStringRect.bottom + 1;
            left := UserWindow^.portRect.left;
            bottom := ConfirmStringPos.v +
            UserWindowFontInfo.descent;
            right := UserWindow^.portRect.right;
        end;

    (Set to false, assume no hook code override)
        UserWindowProcsChanged := F;

    (Allow setup of any windows from the Hooks unit)
        if SetUpHooksWindows = T then
        begin
            ShowWindow(UserWindow);
            (We succeeded! Return T)
            GotWindows := T;
        end
        else
        begin
            (SetUpHooksWindows failed)
            LDispose(NameListHdl);
            CloseWindow(UserWindow);
        end;
    end;

end;

procedure DoDrag;
(This code drags windows around)
begin
    DragWindow(WhichWindow, Evt.where, DragRect);
end;

procedure ActivateUserWindow;
(This code activates the apps windows)
begin
    if UserWindowProcsChanged then
        HooksActivateUserWindowProc (Let hook code do its magic)
    else
        LActivate(T, NameListHdl);
end;

procedure DeactivateUserWindow;
(This code deactivates the apps windows)
begin
    if UserWindowProcsChanged then
        HooksDeactivateUserWindowProc (Let the hook code do its
    magic)
    else
        LActivate(F, NameListHdl);
end;

procedure UpdateNameList;
(This code updates the list of names in the NameListHandle)
var
    theAddress: AddrBlock;
    theCell: Cell;
    theNTT: EntityName;
    theNum: Integer;
    err: OSErr;
    thePtr: Ptr;
begin
    with LookUpNamePb do

```

```

begin
  if NameListLength <> 0 then
    LDelRow(NameListLength, 0, NameListHdl); (Delete
any rows present)
    NameListLength := xMPPpb.numGotten; (Get the new # of
rows needed)
    if NameListLength <> 0 then
      begin
        theNum := LAddRow(NameListLength, 0,
NameListHdl); (Add the # of new rows)
        theCell.h := 0;
        for theNum := 1 to NameListLength do
          begin
            err := NBPEExtract(@LookUpBuffer, NameList-
tLength, theNum, theNTT, theAddress); (extract a name)
            if err = noErr then
              begin
                theCell.v := theNum - 1; (Cells are
numbered starting from zero so bump row # back)
                thePtr := Ptr(ORD4(@theNTT.objStr) + 1);
(Pt to the data)
                LSetCell(thePtr, length(theNTT.objStr),
theCell, NameListHdl); (Set the cells data)
              end;
            end;
            (Select the first row)
            theCell.h := 0;
            theCell.v := 0;
            LSetSelect(T, theCell, NameListHdl);
          end;
        end;
      end;
    end;

  procedure UpdateUserWindow;
  (This code updates the UserWindow)
  var
    theRgn: RgnHandle;
  begin
    if UserWindowProcsChanged then
      HooksUpdateUserWindowProc (Let hook code do its magic)
    else
      begin
        ($IFC TALK_DEBUG )
          ForeColor(redColor);
          FillRect(DebugOnRect, black);

          if SelfSendOn then
            begin
              ForeColor(cyanColor);
              FillRect(SelfSendOnRect, black);
            end
          else
            begin
              ForeColor(cyanColor);
              FrameRect(SelfSendOnRect);
            end;
          end;
          ForeColor(blackColor);
        ($ENDC)

        FrameRect(ListFrameRect); (Put a frame around list)
        theRgn := UserWindow^.visRgn;
        LUpdate(theRgn, NameListHdl); (Update the list)
        DrawLookUpString; (Draw in last LookUpName result)
        DrawConfirmString; (Draw in last ConfirmName result)
      end;
    end;

  procedure DoActivate;
  (This code activates any app windows)
  var
    theWindow: WindowPtr;
  begin
    theWindow := WindowPtr(Evt.message);

```

```

SetPort(theWindow);
if BAnd(ActiveFlag, Evt.modifiers) <> 0 then
  begin
    if theWindow = UserWindow then
      ActivateUserWindow
    else
      ActivateHooksWindow(theWindow); (Let the hook code
activate its windows)
    end
  else
    begin
      if theWindow = UserWindow then
        DeactivateUserWindow
      else
        DeactivateHooksWindow(theWindow); (Let the hook
code deactivate its windows)
      end;
    end;

  procedure DoUpdate;
  (This proc does updates of app windows)
  var
    oldPort: GrafPtr;
    theWindow: WindowPtr;
  begin
    GetPort(oldPort);

    theWindow := WindowPtr(Evt.message);
    SetPort(theWindow);

    BeginUpdate(theWindow);

    if theWindow = UserWindow then
      UpdateUserWindow
    else
      UpdateHooksWindow(theWindow); (Let the hook code update
any of its windows)

    EndUpdate(theWindow);

    SetPort(oldPort);
  end;

  procedure DoMFEvt;
  (code handles MultiFinder events such as deac/activation of)
  (windows upon suspend/resume events)
  var
    theWindow: WindowPtr;
  begin
    theWindow := FrontWindow;
    case BSR(Evt.message, 24) of (high byte of message)
      SUSPENDRESUMEMSG:
        begin
          if BAnd(Evt.message, RESUMEMASK) <> 0 then
            begin
              SleepTime := 10;
              SetPort(theWindow);
              if theWindow = UserWindow then
                ActivateUserWindow
              else
                ActivateHooksWindow(theWindow); (Let the
hook code activate its windows)
            end
          else
            begin
              SleepTime := 60;
              SetPort(theWindow);
              if theWindow = UserWindow then
                DeactivateUserWindow
              else
                DeactivateHooksWindow(theWindow); (Let the
hook code deactivate its windows)
            end;
          end;
        end;
    end;
  end;

```

```

        otherwise
        end;
    end;

function UserRegistered: Boolean;
(This proc lets User register on the network, it return T)
(T if it succeeds)
    label
    100;
var
    localATPPb: ATPParamBlock;
    theDialog: DialogPtr;
    theDRec: DialogRecord;
    oldPort: GrafPtr;
    item: Handle;
    itemHit: Integer;
    localMPPPB: MPPParamBlock;
    err: OSErr;
    itemRect: Rect;
    theStr32Hdl: Str32Hdl;
    theString: Str255;
begin
    (Assume we fail)
    UserRegistered := F;

    (Get the string entered in the 'Chooser')
    theStr32Hdl := Str32Hdl(GetResource('STR ', rCH00SERUSER-
STRID));

    (If there is no default user name from the chooser, then set
theString to the)
    (null string)
    if theStr32Hdl = nil then
        theString := '';

        GetPort(oldPort);

        theDialog := GetNewDialog(rUSERNAMEDLOGID, @theDRec,
WindowPtr(-1));
        SetPort(theDialog);

        ShowWindow(theDialog);

100:
    if theStr32Hdl <> nil then
        begin
            HLock(Handle(theStr32Hdl));
            theString := theStr32Hdl^; (theString = the chooser
string)
            HUnlock(Handle(theStr32Hdl));
        end;

        (Show theString in the dialog and select it)
        GetDItem(theDialog, rUSERNAMEITEM, itemHit, item,
itemRect);
        SetIText(item, theString);
        SelIText(theDialog, rUSERNAMEITEM, 0, 32767);

        (Loop until OK or Cancel is hit)
        repeat
            ModalDialog(nil, itemHit)
        until ((itemHit = ok) or (itemHit = cancel));

        if itemHit = ok then
            begin
                GetDItem(theDialog, rUSERNAMEITEM, itemHit, item,
itemRect);
                GetIText(item, theString);
                itemHit := length(theString);
                if (itemHit > 0) and (itemHit <= MAXNAMELENGTH) then
                    begin
                        (theString must be > 0 but < 32)
                        (First try to open a socket)
                        with localATPPb do

```

```

begin
    ioCompletion := nil;
    atpSocket := 0;
    with addrBlock do
        begin
            aNet := 0;
            aNode := 0;
            aSocket := 0;
        end;
    end;

    err := POpenATPSkt(@localATPPb, SYNC);
    if err = noErr then
        begin
            UserSkt := localATPPb.atpSocket; (Save the
socket #)
            (Try to register the user on the network)
            NBPSetsEntity(@UserNTT, theString, USERTYPE,
ANYZONE);
            NBPSetsNTE(@NBPsNTE, UserNTT.objStr, USER-
TYPE, ANYZONE, UserSkt); (Set up the nte that NBP wants)

            with localMPPPB do
                begin
                    ioCompletion := nil;
                    interval := 3;
                    count := 3;
                    entityPtr := @NBPsNTE;
                    verifyFlag := DOVERIFY; (Make sure we
register with a unique name)
                end;

                err := PRegisterName(@localMPPPB, SYNC);
                if err = noErr then
                    begin
                        (We registered successfully!)
                        UserRegistered := HooksRegistered; (Let
the hook code do its magic)

                        ($IFC TALK_DEBUG)
                        (Enable self sending if the new calls exist)
                        if NewCallsExist then
                            begin
                                localMPPPB.newSelfFlag := SENDSELF;
                                err := PSetSelfSend(@localMPPPB,
SYNC);

                                if err = noErr then
                                    begin
                                        OldSelfFlag :=
localMPPPB.oldSelfFlag; (Save the old self flag state)
                                        SelfSendOn := T; (Mark our
selfsend flag as on)
                                    end
                                else
                                    SelfSendOn := F; (We failed, mark
our selfsend flag as off)
                                end
                            end
                        else
                            SelfSendOn := F; (Mark our selfsend
flag as off)
                        ($ENDC)

                        end
                    else
                        begin
                            (PRegisterName failed)
                            (Close the UserSkt)
                            localATPPb.atpSocket := UserSkt;
                            err := PCloseATPSkt(@localATPPb, SYNC);
                        end;
                    end;
                end
            end
        else
            (User didn't enter a valid string)
            goto 100;
        end;
    end;
end;

```

```

end;

(Close the dialog and return the result)
CloseDialog(theDialog);
SetPort(oldPort);
end;

function AppleTalkOK: Boolean;
(This proc makes sure we're running on at least a 512KE)
(, it returns T if so and checks to see if the new calls
exist)
var
    err: OSErr;
    theWorld: SysEnvRec;
begin
    (Assume we failed)
    AppleTalkOK := F;

    err := MPPOpen; (Open AppleTalk)
    if err = noErr then
        begin
            err := SysEnvirons(1, theWorld);
            if err = noErr then
                begin
                    with theWorld do
                        begin
                            if machineType >= env512KE then
                                begin
                                    (We've got at least a 512KE)
                                    if atDrvVrsNum >= XNCVERSION then
                                        (We've got the new calls)
                                        NewCallsExist := T;
                                        AppleTalkOK := UserRegistered; (Let the
user try to register)
                                end;
                            end;
                        end;
                    end;
                end;
            end;
        end;

    procedure AppleTalkCallChecks;
    (This proc checks to see if any of ASYNC calls have fin-
ished)
    var
        oldPort: GrafPtr;
    begin
        with LookUpNamePb do
            begin
                if CallDone <> 0 then
                    begin
                        PbInUse := F;
                        CallDone := 0;
                        GetPort(oldPort);
                        SetPort(UserWindow);
                        UpdateNameList;
                        LookUpString := 'done';
                        DrawLookUpString;
                        SetPort(oldPort);
                    end;
                end;
            end;
        end;

    (Allow any AppleTalk call checks in the Hooks unit)
    HooksAppleTalkCallChecks;
end;

procedure CloseUpAppleTalk;
(This proc closes up the apps AppleTalk socket lets hook)
(code cleanup anything it did with AppleTalk)
var
    localATPPb: ATPParamBlock;
    localMPPPb: MPPParamBlock;
    err: OSErr;
begin
    (A LookUp call may not have finished, if new calls exist)

```

```

(then KillNBP it, if not then we must wait till it fin-
ishes.)
with LookUpNamePb do
    begin
        if PbInUse then
            begin
                (There is a LookUpName in progress)
                if NewCallsExist then
                    begin
                        (We've got the new calls so we can kill it)
                        with localMPPPb do
                            begin
                                ioCompletion := nil;
                                nKillQE1 := @LookUpNamePb.xMPPPb;
                            end;
                        err := PKillNBP(@localMPPPb, SYNC);
                        if err <> noErr then
                            begin
                                (We failed to KillNBP the LookUp so we'll just loop till
the LookUp is done)
                                (NOTE that we test the CallDone field since its changed by
the completion routine, NOT)
                                (the PbInUse field)
                                while LookUpNamePb.CallDone = 0 do
                                    ;
                                end;
                            end;
                        else
                            (The new calls dont exist so we'll just loop till the
LookUp is done)
                            (NOTE that we test the CallDone field since its changed by
the completion routine, NOT)
                            (the PbInUse field)
                            while LookUpNamePb.CallDone = 0 do
                                ;
                            end;
                        end;
                    end;
                end;
            end;
        end;

    (Allow Hooks unit to closeup any AppleTalk calls it made)
    HooksCloseUpAppleTalk;

    (Close the UserSkt)
    localATPPb.atpSocket := UserSkt;
    err := PCloseATPSkt(@localATPPb, SYNC);
    if err <> noErr then
        SysBeep(1);

    (Remove the users name from the network)
    localMPPPb.entityPtr := @UserNTT;
    err := PRemoveName(@localMPPPb, SYNC);
    if err <> noErr then
        SysBeep(1);

    ($IFC TALK_DEBUG)
    (Restore the SelfSendFlag to its former state)
    if NewCallsExist then
        begin
            if SelfSendOn then
                begin
                    if OldSelfFlag = 0 then
                        begin
                            (Turn it off if it wasn't on before)
                            localMPPPb.newSelfFlag := OldSelfFlag;
                            err := PSetSelfSend(@localMPPPb, SYNC);
                            end;
                        end;
                    end;
                end;
            end;
        end;

    ($ENDC)
end;

function AppleTalkGlobalsSetUp: Boolean;
(This proc inits any app AppleTalk globals)
begin
    (Assume error)

```



```

AppleTalkGlobalsSetup := F;

with LookUpNamePb do
begin
    PbInUse := F;
    CallDone := 0;
end;

ConfirmString := 'never done';
LookUpString := 'never done';

(Allow setup of AppleTalk globals from the Hooks unit)
if HooksAppleTalkGlobalsSetup then
    AppleTalkGlobalsSetup := T;
end;

procedure DoWNEs;
(Do a few WNEs to make sure our dialog comes up in front)
var
    theResult: Boolean;
begin
    theResult := WaitNextEvent(everyEvent, Evt, 0, nil);
    theResult := WaitNextEvent(everyEvent, Evt, 0, nil);
    theResult := WaitNextEvent(everyEvent, Evt, 0, nil);
    theResult := WaitNextEvent(everyEvent, Evt, 0, nil);
    theResult := WaitNextEvent(everyEvent, Evt, 0, nil);
end;

procedure DoAbout;
(This proc shows our About box)
var
    theDialog: DialogPtr;
    theDRec: DialogRecord;
    oldPort: GrafPtr;
    itemHit: Integer;
begin
    GetPort(oldPort);

    theDialog := GetNewDialog(rABOUTDLOGID, @theDRec, Win-
dowPtr(-1));
    SetPort(theDialog);
    ShowWindow(theDialog);

    repeat
        ModalDialog(nil, itemHit)
    until (itemHit = ok);

    CloseDialog(theDialog);

    SetPort(oldPort);
end;

procedure DoAppleMenu (item: Integer);
(This proc handles the Apple Menu)
begin
    case item of
        ABOUTITEM:
            DoAbout;
        otherwise
            end;
    end;
end;

procedure DoFileMenu (item: Integer);
(This proc handles the File Menu)
begin
    case item of
        QUITITEM:
            DoneFlag := T;
        otherwise
            end;
    end;
end;

procedure DoLookUp;
(This proc does an ASYNC LookUpName)

```

```

var
    oldPort: GrafPtr;
    err: OSErr;
begin
    with LookUpNamePb do
        begin
            if PbInUse = F then
                begin
                    (If we're not already doing one)
                    NBPSetEntity(@LookUpNTT, ANYOBJ, USERTYPE,
ANYZONE); (Setup who to look for)

                    CallDone := 0;

                    with xMPPPB do
                        begin
                            ioCompletion := @XCompletionRoutine;
($IFC TALK_DEBUG )
                                interval := 2;
                                count := 3;
($ELSC)
                                interval := 6;
                                count := 5;
($ENDC)
                                entityPtr := @LookUpNTT;
                                retBuffPtr := @LookUpBuffer;
                                retBuffSize := sizeof(LookUpBuffer);
                                maxToGet := MAXTOLOOKUP;
                        end;

                        err := PLookUpName(@xMPPPB, ASYNC);
(Show the LookUps progress)
                        GetPort(oldPort);
                        SetPort(UserWindow);
                        if err = noErr then
                            begin
                                PbInUse := T;
                                LookUpString := 'in progress';
                                end
                            else
                                LookUpString := 'error';
                                DrawLookUpString;
                                SetPort(oldPort);
                                end;
                        end;
                    end;
                end;
            end;

        procedure DoConfirm;
        (This proc does a SYNC ConfirmName)
        var
            theAddress: AddrBlock;
            nameIsSelected: Boolean;
            theCell: Cell;
            extractedNTT, ntt2Confirm: EntityName;
            oldPort: GrafPtr;
            err: OSErr;
        begin
            if LookUpNamePb.PbInUse then
                begin
                    (Dont try to ConfirmName while a LookUp is in progress)
                    GetPort(oldPort);
                    SetPort(UserWindow);
                    ConfirmString := 'LookUp in progress';
                    DrawConfirmString;
                    SetPort(oldPort);
                end
            else
                begin
                    theCell.h := 0;
                    theCell.v := 0;
                    nameIsSelected := LGetSelect(T, theCell,
NameListHdl); (Get the selected list item)

                    if nameIsSelected then

```

```

begin
  (Only do a Confirm if an item in the list is selected)
  theCell.v := theCell.v + 1;
  err := NBPEExtract(@LookUpBuffer, NameListLength,
theCell.v, extractedNTT, theAddress); (Get the address)
  if err = noErr then
    begin
      NBPSetsEntity(@ntt2Confirm,
extractedNTT.objStr, extractedNTT.typeStr,
extractedNTT.zoneStr);

      err := NTTEExists(ntt2Confirm, theAddress);
    (Check if the NTT exists)
    (Show the result of the ConfirmName)
    GetPort(oldPort);
    SetPort(UserWindow);
    if err = noErr then
      ConfirmString := 'confirmed'
    else
      begin
        case err of
          nbpNoConfirm:
            ConfirmString := 'can''t confirm ';
          nbpConfDiff:
            ConfirmString := 'moved';
          otherwise
            ConfirmString := 'error';
        end;
        DrawConfirmString;
        SetPort(oldPort);
      end;
    end;
  end;
end;

procedure DoSendRequest;
(This is the generic codes response to choosing 'Send')
begin
  SysBeep(1);
end;

procedure DoSpecialMenu (item: Integer);
(This proc handles the Special Menu)
begin
  case item of
    LOOKUPITEM:
      DoLookUp;
    CONFIRMITEM:
      DoConfirm;
    SENDITEM:
      DoSendRequest;
    otherwise
      end;
  end;
end;

procedure DoMenu (menuChoice: LongInt);
(This proc handles menu selections)
var
  theItem, theMenu: Integer;
begin
  theItem := LoWord(menuChoice);
  theMenu := HiWord(menuChoice);

  if HookedMenuChoice(theMenu, theItem) = F then
    begin
      (Hook code didn't handle menu choice so generic code must)
      case theMenu of
        APPLMENUID:
          DoAppleMenu(theItem);
        FILEMENUID:
          DoFileMenu(theItem);
        SPECIALMENUID:
          DoSpecialMenu(theItem);

```

```

      otherwise
        end;
      end;

      HiliteMenu(0);
    end;

    procedure DoKDown;
    (This code handles keydowns)
    type
      Trick = packed record
        case Boolean of
          TRUE: (
            long: LongInt
          );
          FALSE: (
            chr3, chr2, chr1, chr0: Char
          )
        end;
    var
      charCode: Char;
      trickVar: Trick;
    begin
      trickVar.long := Evt.message;
      charCode := trickVar.chr0;
      (IF BitAnd(Evt.modifiers, CmdKey) = CmdKey THEN)
      if BAnd(CmdKey, Evt.modifiers) <> 0 then
        DoMenu(MenuKey(charCode));
      end;

      procedure DoContentHit;
      (This code handles hits in the apps windows)
      var
        clickResult: Boolean;
        localPt: Point;
      begin
        if WhichWindow <> FrontWindow then
          SelectWindow(WhichWindow)
        else
          begin
            localPt := Evt.where;
            SetPort(WhichWindow);
            if WhichWindow = UserWindow then
              begin
                if UserWindowProcsChanged then
                  HooksUserWindowDoContentHit (Let the hook code
do its magic)
              end
            else
              begin
                GlobalToLocal(localPt);
                if PtInRect(localPt, ListFrameRect) then
                  clickResult := LClick(localPt,
Evt.modifiers, NameListHdl);
                end;
              end
            else
              (A hook window was hit, let it handle it)
              DoHooksWindowContentHit;
            end;
          end;

          procedure DoMDown;
          (This code handles mouse down events)
          var
            theResult: Integer;
          begin
            theResult := FindWindow(Evt.where, WhichWindow);
            case theResult of
              inDrag:
                DoDrag;
              inMenuBar:
                DoMenu(MenuSelect(Evt.where));
              inContent:
                DoContentHit;

```

```

        otherwise
        end;
    end;
end;

procedure MainLoop;
(This is the apps event loop)
var
    theResult: Boolean;
begin
    while DoneFlag = F do
    begin
        theResult := WaitNextEvent(everyEvent, Evt, Sleep-
Time, nil);
        AppleTalkCallChecks; (Check to see if any ASYNC calls
completed)
        if theResult then
        begin
            (Handle any event)
            case Evt.what of
                keyDown:
                    DoKDown;
                mouseDown:
                    DoMDown;
                activateEvt:
                    DoActivate;
                updateEvt:
                    DoUpdate;
                MFEVENT:
                    DoMFEvent;
                otherwise
            end;
        end;
    end;
end;

($I-)
(We do our own inits)
begin
    SetUpToolbox;
    SetUpGlobals;
    DoWNEs;

    if AppleTalkOK then
    begin
        SetUpMenus;

        if GotWindows then
        begin
            if AppleTalkGlobalsSetup then
            begin
                MainLoop;

                CloseWindows;
            end;
        end;

        CloseUpAppleTalk;
    end;

    ExitToShell;
end.

```

Listing: resources.r

```

resource 'SIZE' (-1) (
    dontSaveScreen,
    acceptSuspendResumeEvents,
    disableOptionSwitch,
    canBackground,
    multiFinderAware,
    81920,
    81920

```

```

);

resource 'DITL' (128, purgeable, preload) (
    /* array DITLarray: 3 elements */
    /* [1] */
    (130, 134, 150, 213),
    Button (
        enabled,
        "Continue"
    ),
    /* [2] */
    (8, 63, 26, 298),
    StaticText (
        disabled,
        "AACK - Version 1.1 - by Ajay Nath"
    ),
    /* [3] */
    (56, 80, 74, 266),
    StaticText (
        enabled,
        "Copyright © 1989 Ajay Nath"
    )
);

resource 'DITL' (129, purgeable, preload) (
    /* array DITLarray: 4 elements */
    /* [1] */
    (89, 23, 109, 83),
    Button (
        enabled,
        "OK"
    ),
    /* [2] */
    (90, 250, 110, 310),
    Button (
        enabled,
        "Cancel"
    ),
    /* [3] */
    (6, 10, 41, 319),
    StaticText (
        disabled,
        "Enter a name (31 characters or less) to "
        "use on the network"
    ),
    /* [4] */
    (52, 16, 72, 317),
    EditText (
        enabled,
        ""
    )
);

resource 'DLOG' (128, purgeable, preload) (
    (36, 146, 204, 490),
    dBoxProc,
    invisible,
    noGoAway,
    0x0,
    128,
    ""
);

resource 'DLOG' (129, purgeable, preload) (
    (36, 150, 160, 482),
    dBoxProc,
    invisible,
    noGoAway,
    0x0,
    129,
    ""
);

```

```

resource 'MENU' (128, preload) (
    128,
    textMenuProc,
    0x7FFFFFFD,
    enabled,
    apple,
    ( /* array: 2 elements */
        /* [1] */
        "About AACK...", noIcon, "", "", plain,
        /* [2] */
        "-", noIcon, "", "", plain
    )
);

```

```

resource 'MENU' (129, preload) (
    129,
    textMenuProc,
    allEnabled,
    enabled,
    "File",
    ( /* array: 1 elements */
        /* [1] */
        "Quit", noIcon, "Q", "", plain
    )
);

```

```

resource 'MENU' (130, preload) (
    130,
    textMenuProc,
    0x7FFFFFFD,
    enabled,

```

```

    "Edit",
    ( /* array: 6 elements */
        /* [1] */
        "Undo", noIcon, "Z", "", plain,
        /* [2] */
        "-", noIcon, "", "", plain,
        /* [3] */
        "Cut", noIcon, "X", "", plain,
        /* [4] */
        "Copy", noIcon, "C", "", plain,
        /* [5] */
        "Paste", noIcon, "V", "", plain,
        /* [6] */
        "Clear", noIcon, "", "", plain
    )
);

```

```

resource 'MENU' (131, preload) (
    131,
    textMenuProc,
    allEnabled,
    enabled,
    "Special",
    ( /* array: 3 elements */
        /* [1] */
        "LookUp", noIcon, "L", "", plain,
        /* [2] */
        "Confirm", noIcon, "F", "", plain,
        /* [3] */
        "Send", noIcon, "S", "", plain
    )
);

```



Color Workshop

Capture Color Icons



Steve and Patricia Sheets
Herdon, VA
Volume 5, Number 12

Color Icons: Editor and cIcnCapture

This month's Color Quickdraw column consists of less talk and more code. In the last column, the building block data structures needed in order to use and understand color icons were discussed. A new Rez/Derez resource source template was given that allowed easier creation of 'cIcn' resource (Color Icon template resource). While the template is helpful in learning and understanding color icons, and is an excellent way to present color icons on the written page, most users would prefer to use some graphic base tool to create a color icon. This column contains a quick review of the two color icon editors freely available to the development community. Then, the column will present this month's source code, 'cIcnCapture', a FKEY utility designed to capture an image on the screen and convert it to a color icon resource.

Public Domain Color Icon Editors

There are two different color icon editors available in Public Domain. The very first editor released was the application 'CIcon Edit' by Ben Haller of AppleSauce Designs. 'CIcon Edit' allows the users to edit color icons of 32 by 32 pixels size that have up to 16 separate colors. The editor can import a 'cIcn' resource, a 'ICON' resource or a 'ICN#' resource. This feature makes the tool exceptionally good at coloring existing icon art work. 'CIcon Edit' itself is the easiest to use, with a good command set to manipulate the image (move, roll, flip, rotate). This tool has been around the longest (latest version I've seen is 1.1) and is virtually bug proof. Unfortunately, the application is lacking in certain areas. This editor is unable to edit a color icon that is any other size than 32 by 32 pixels and 16 colors. There is no method of adjusting any other settings of the resource. 'CIcon Edit' does not have a palette of tools (ala MacPaint); the user draws the image a pixel at a time. Also, the load command does not give a list of all the 'cIcn' resources of the target file. Thus, the user has to keep track of all 'cIcn' and their ID numbers himself. To be really productive with this tool, the user needs to work ResEdit.

The other PD color icon editor solves this problem by being a ResEdit extension. 'ResWare' by Frédéric Miserey of None Corp in Paris consists of certain resources that the user copies into his ResEdit application (replacing some existing ones). Then the user can use ResEdit directly to observe, create and edit 'cIcn' resources. This utility is a more powerful editor than 'CIcon Edit'. 'ResWare' has a complete tool palette that includes pen, eraser, fill bucket, lasso, and other rectangle commands. The editor can use single color patterns when drawing. There is no practical limit to the size of the color icon, nor the number of colors. Also the editor allows the user to change other portions

of the 'cIcn' resource including horizontal and vertical size, component count, component size and pixmap method (Chunky, Chunky/Planar or Planar). However the editor is still in beta form (latest version I've seen is 1.0b4) and has caused numerous crashes. 'ResWare' has no import feature, and the editor, while more powerful, is not as easy to use for beginners as 'CIcon Edit'.

Importing Images Into Color Icons

While both editors are useful in creating color icons, they both suffer from a common fault. They do not allow easy import of an image on the screen into a 'cIcn' resource. At first, it may not seem as if this is a problem. Of course, the editor will only be used to create a 'cIcn' resource. However, this is not always the case. Look at what happened with original Quickdraw and the creation of 'ICON' and 'ICN#' resources. Even though there were a few icon editors, many non-technical users created their images with the more powerful Paint and Draw programs. Then they handed the complete art work to a programmer, expecting him to be able to use it directly. The programmer had no way to directly transfer an art document into a 'ICON' or 'ICN#' resource. In many cases, the programmer had to reenter the image into the icon editor one pixel at a time. To solve this problem, many programmers created various Icon-Grapper types of utilities. These utilities, usually Desk Accessories or FKEYs, allowed the programmer to capture any portion of the screen and convert it into an icon resource. Since it captured an image on the screen and did not need to directly convert any type of document, these utilities would work with any Paint or Draw application.

A user who wanted to convert a Pixel Paint or MacDraw // image into a 'cIcn' resource would have the identical problem. The problem is further complicated, since the user would have to worry not only about the 32 by 32 pixel image, but would have to import a color list, a mask and a black and white icon. I had this problem. The graphic designer of my latest project presented the technical team with a group of Pixel Paint files, each one having a dozen or so images on it. It would have taken hours of work to reenter the images using one of the color icon editors. That estimate was assuming that the artwork was the finished product, and not going to be revised. This started looking like a potential nightmare.

cIcnCapture

The solution to this problem is being provided as the sample source code. 'cIcnCapture' is an FKEY utility that can capture a 32 by 32 pixel image of the screen, containing up to 256 different colors, and create a 'cIcn' resource that contains this image. The code uses the Quickdraw call, GetCPixel, to deter-

mine the color of each pixel in the image. Since the code uses this call instead of reading the screen Pixmap directly, it will work on any depth or type of screen or monitor as long as Color Quickdraw is installed. Once the image has been captured, the information is saved out to a specified resource file.

Using this utility on our software project, it took a few minutes to import all of the color images into 'cicn' resource. 'CIcon Edit' and 'ResEdit' were used afterward, but mainly to make sure the images had the correct ID number, and that they were centered on the 32 by 32 pixel icon.

Next Column

Anyone remember the game, 'Daleks'? This application was one of the first, and best, Public Domain games around. Next issue's sample code will be 'Color Daleks'. The program will show how to directly draw and manipulate color icons. Various color drawing modes will be explained, as well as some animation speed up tricks.

```
( cicnCapture- Color Icon Capture FKEY )
( by Steve Sheets for MacTutor 9/3/89 )
( FKEY is normal saved to resource type 'FKEY', ID = 7 )
( This FKEY captures a 32 by 32 Pixel section of the screen
and converts it into a 'cicn' resource (Color Icon )
( template). When the user activates the Function Key and
click somewhere on the screen, that portion of the )
( screen is recorded (colors and pixels) and a cicn handle is
created. Remember a cicn handle is different than )
( an actual Color Icon handle. The cicn handle/resource is a
template for a Color Icon. The Color Quickdraw )
( command, GetCIcon, uses the cicn resource to determine how
to make a Color Icon. )
( FKEY is written in Think's Lightspeed Pascal 2.0. )
```

unit cicnCapture;

interface

uses

Quickdraw, ToolIntf;

(FKEY (function key) code resources must have be an unit with a single externally reference procedure called)
('Main'.)

procedure Main;

implementation

procedure Main;

(The 'cicn' resource consists of three variable length parts. The first part contains the Pixmap info, the Black)
(& White Icon and the Mask. The second part contains the Color Lookup table (ie. list of colors used in the)
(cicn). The size of this part is dependent on the number of different colors in the cicn. The last part of the)
(resource is the pixel data. The size of this is dependent on the number of different colors in the cicn. If)
(there are 2 to 16 colors, the pixel info can be stored in a 4 bit pixel map, if the number of colors is 17 to)
(256, then the info must be stored in a 8 bit pixel map. Given the fact this FKEY only captures 32 by 32 cicn,)
(the size of the first part is fixed, while the size of the third part is one of two possible fixed sizes.)

type

IconData = packed array[0..31] of LongInt;

(First part of cicn resource for a 32 by 32 pixel cicn.)

```
cicnData = record
  baseAddr: LongInt;
  rowbytes: INTEGER;
  bounds: Rect;
  pmVersion: INTEGER;
  packType: INTEGER;
  packSize: LongInt;
  hRes: LongInt;
  vRes: LongInt;
  pixelType: INTEGER;
  pixelSize: INTEGER;
  cmpCount: INTEGER;
  cmpSize: INTEGER;
  planeBytes: LongInt;
  pmTable: LongInt;
  pmReserved: LongInt;

  MaskBaseAddr: LongInt;
  MaskRowBytes: INTEGER;
  MaskBounds: Rect;
```

```
BMapBaseAddr: LongInt;
BMapRowBytes: INTEGER;
BMapBounds: Rect;
```

IconData: LongInt;

theMask: IconData;

theBMap: IconData;

```
ctSeed: LongInt;
ctFlags: INTEGER;
ctSize: INTEGER;
```

end;

cicnPtr = ^cicnData;

(2nd part (CLUT) of cicn resource for 32 by 32 pixel cicn.)

```
CTableData = packed array[0..255] of ColorSpec;
CTablePtr = ^CTableData;
```

(3rd part of cicn resource for 8 bit, 32 by 32 pixel cicn.)

```
pixel8Data = packed array[0..31, 0..31] of 0..255;
pixel8Ptr = ^pixel8Data;
```

(3rd part of cicn resource for 4 bit, 32 by 32 pixel cicn.)

```
pixel4Data = packed array[0..31, 0..15] of 0..255;
pixel4Ptr = ^pixel4Data;
```

var

```
thePos: Point;
thePixel8: pixel8Data;
theNumColors: INTEGER;
theIcon: IconData;
theColorList: packed array[0..255] of RGBColor;
theCICN: Handle;
```

(Tell us if Color Quickdraw is installed on machine.)

function IsColorQuickdraw: BOOLEAN;

const

```
ROM85Loc = $28E;
TwoHighMask = $C000;
```

type

WordPtr = ^INTEGER;

var

Wd: WordPtr;

```

begin
  Wd := POINTER(ROM85Loc);
  IsColorQuickdraw := (BitAnd(Wd, TwoHighMask) = 0);
end;

( Sets cursor to show an upper left portion of a square.)

procedure InitSquare;
var
  count: INTEGER;
  Square: Cursor;
begin
  with Square do
  begin
    data[0] := $FFFF;
    mask[0] := 0;
    for count := 1 to 15 do
    begin
      data[count] := $8000;
      mask[count] := 0;
    end;
    hotSpot.v := 0;
    hotSpot.h := 0;
  end;
  SetCursor(Square);
end;

( Wait until the user presses the mouse, making sure the mouse
was not pressed to begin with. Then stores )
( the position of the mouse and flushes the mouse events. )

procedure GetPos;
begin
  while button do
  ;
  while not button do
  ;
  GetMouse(thePos);
  FlushEvents(mDownMask + mUpMask, 0);
end;

( Given the position of the mouse, captures a 32 by 32 array
of pixels and a list of the colors. The )
( pixel information is not stored as RGB values, but as a
number (zero count) that represents that )
( RGB value in the Color List (ie. value of 2 equals third
color in list). Thus every time a pixel on )
( the screen is looked at with the GetCPixel command, that RGB
value is looked for in the Color List.)
( If the color is found, that number is stored in the 2
dimensional pixel array. If the pixel uses a )
( new RGB color (not found in the list), that color is added
to the list. The number that represents )
( that new color is stored in the pixel array. This way, at
the end, the program knows exactly what )
( colors are used. The code does not care about the depth of
the screen (1, 2, 4, 8, 16 or 24 bits )
( per pixels). However this cicon resource uses the Chunky
model, thus only 256 colors can be used )
( (fairly safe limit). The list will always contain black and
white as the first two colors in the list.)
( This procedure also creates a black and white 32 by 32 bit
icon for the mask and the Black and )
( White icon portion of the cicon. Any color other than white
indicates a black bit. )

procedure GetPixels;
var
  h, v, theNum, count: INTEGER;
  theColor: RGBColor;
begin
  theNumColors := 1;
  with theColorList[0] do
  begin

```

```

    red := $FFFF;
    green := $FFFF;
    blue := $FFFF;
  end;
  with theColorList[1] do
  begin
    red := 0;
    green := 0;
    blue := 0;
  end;
  for v := 0 to 31 do
  begin
    theIcon[V] := 0;
    for h := 0 to 31 do
    begin
      GetCPixel(thePos.h + h, thePos.v + v, the-
eColor);
      theNum := -1;
      for count := 0 to theNumColors do
      if (theColor.red = theColorList[count].red)
then
        if (theColor.green =
theColorList[count].green) then
          if (theColor.blue =
theColorList[count].blue) then
            theNum := count;
            if (theNum = -1) and (theNumColors < 255) then
            begin
              theNumColors := theNumColors + 1;
              theNum := theNumColors;
              theColorList[theNum] := theColor;
            end;
            thePixel8[v, h] := (theNum mod 256);
            if (theColor.red <> -1) or (theColor.green <>
-1) or (theColor.blue <> -1) then
              BitSet(@theIcon[V], h);
            end;
          end;
        end;
      end;
    end;
  end;

( Having pixel infor, allocates memory for cicon handle.)

procedure MakeCICN;
var
  theCICNPtr: ciconPtr;
  theCTablePtr: CTablePtr;
  thePixel8Ptr: Pixel8Ptr;
  the4PixelPtr: Pixel4Ptr;
  theRect: Rect;
  count, v, h, theTableSize, theImageRowBytes, theB-
itsPixel, thePixelDataSize: INTEGER;
begin
  SetRect(theRect, 0, 0, 32, 32);

  ( If number of colors is more than 16, then use 8 bit pixel
map, else use 4 bit pixelmap (Rowbytes of pixmap is )
( different, bit depth is different, size of pixel data is
different). Notice that Rowbytes must have the high bit )
( set.)

  if (theNumColors > 16) then
  begin
    theBitsPixel := 8;
    theImageRowBytes := $8020;
    thePixelDataSize := SizeOf(pixel8Data);
  end
  else
  begin
    theBitsPixel := 4;
    theImageRowBytes := $8010;
    thePixelDataSize := SizeOf(pixel8Data) div 2;
  end;
  theTableSize := (theNumColors + 1) * SizeOf(ColorSpec);

```

(Allocate size of handle (dependent on number of colors & 4 or 8 bit pixmap.)

```
theCICN := NewHandle(SizeOf(cicnData) + theTableSize +
thePixelDataSize);
HLock(theCICN);
theCICNptr := POINTER(theCICN);
```

(Stuff first part of cicn handle.)

```
with theCICNptr^ do
begin
  baseAddr := 0;
  rowbytes := theImageRowBytes;
  bounds := theRect;
  pmVersion := 0;
  packType := 0;
  packSize := 0;
  hRes := $00480000;
  vRes := $00480000;
  pixelType := 0;
  pixelSize := theBitsPixel;
  cmpCount := 1;
  cmpSize := theBitsPixel;
  planeBytes := 0;
  pmTable := 0;
  pmReserved := 0;
```

```
MaskBaseAddr := 0;
MaskRowBytes := 4;
MaskBounds := theRect;
```

```
BMapBaseAddr := 0;
BMapRowBytes := 4;
BMapBounds := theRect;
```

```
IconData := 0;
```

```
theMask := theIcon;
```

```
theBMap := theIcon;
```

```
ctSeed := 0;
ctFlags := 0;
ctSize := theNumColors;
```

```
end;
```

(Stuff second part, the Color Look Up table (color list).)

```
theCTablePtr := POINTER(ORD4(theCICNptr) + SizeOf(CICNData));
```

```
for count := 0 to theNumColors do
  with theCTablePtr^[count] do
    begin
      value := count;
      rgb := theColorList[count];
    end;
```

(Stuff the third part of the cicn handle, the actual pixmap data. If an 8 bit pixmap is used, the format of)
 (the data is identical to the array used to store the pixel information so the data is simply moved into the)
 (handle. If the pixmap is a 4 bit one, the pixel info is moved into another array that has the format of a 4)
 (bit pixmap data, then that data moved into the handle.)

```
if (theNumColors > 15) then
begin
  thePixel8Ptr := POINTER(ORD4(theCTablePtr) +
theTableSize);
  thePixel8Ptr^ := thePixel8;
end
else
begin
```

```
the4PixelPtr := POINTER(ORD4(theCTablePtr) +
theTableSize);
for v := 0 to 31 do
  for h := 0 to 15 do
    the4PixelPtr^[v, h] := ((thePixel8[v, (h * 2)]
* 16) + thePixel8[v, (h * 2) + 1]) mod 256;
  end;
```

```
HUnlock(theCICN);
end;
```

(Given a cicn handle, store it into a resource file (new or existing one) as a 'cicn' resource.)

```
procedure SaveCICN;
```

```
var
```

```
theWhere: Point;
theRef, theVol: INTEGER;
theE: OSErr;
theReply: SFRReply;
```

```
begin
```

```
theWhere.h := 40;
```

```
theWhere.v := 40;
```

```
SFPutFile(theWhere, 'Save Color Icon to File:', 'cicn
File', nil, theReply);
```

```
if theReply.good then
```

```
begin
```

```
theE := GetVol(nil, theVol);
```

```
theE := SetVol(nil, theReply.vRefNum);
```

```
theRef := OpenResFile(theReply.fName);
```

```
if theRef = -1 then
```

```
begin
```

```
CreateResFile(theReply.fName);
```

```
theRef := OpenResFile(theReply.fName);
```

```
end;
```

```
if theRef <> -1 then
```

```
begin
```

```
AddResource(theCICN, 'cicn', UniqueID('cicn'), '');
```

```
WriteResource(theCICN);
```

```
ReleaseResource(theCICN);
```

```
theCICN := nil;
```

```
CloseResFile(theRef);
```

```
end;
```

```
theE := SetVol(nil, theVol);
```

```
end;
```

```
if theCICN <> nil then
```

```
begin
```

```
DisposeHandle(theCICN);
```

```
SysBeep(1);
```

```
end;
```

```
end;
```

(Main Procedure. If there is Color Quickdraw, init the cursor to a square, get the position of the mouse when the)
 (user clicks, the pixels at that position, convert pixels into a cicn handle, save the cicn handle to a resource and)
 (finally reset the cursor back to an arrow. If there is not Color Quickdraw on the machine, complain!)

```
begin
```

```
if IsColorQuickdraw then
```

```
begin
```

```
InitSquare;
```

```
GetPos;
```

```
GetPixels;
```

```
MakeCICN;
```

```
SaveCICN;
```

```
InitCursor;
```

```
end
```

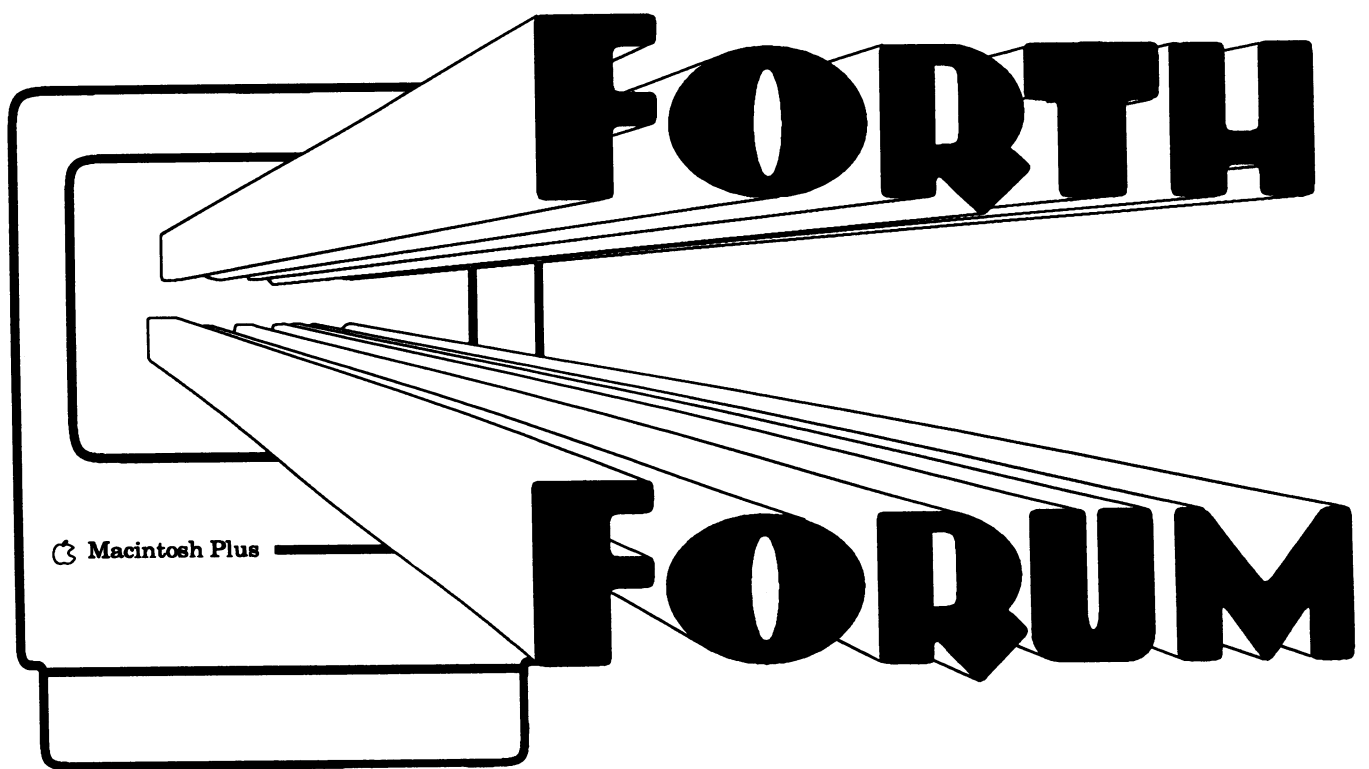
```
else
```

```
SysBeep(1);
```

```
end;
```

```
end.
```





"Looking at system queues"

The utility that I'm going to describe this month can be used to look at various queues of the Macintosh operating system. I was always curious to know what's going on in the background of my machine while I'm working away; that's how this example came to be. At the same time, the program contains another example of popup menus (for which the correct trap call has been added in Mach2.14), and how to use the *userData* field in the user variable area to display messages in a task window.

A number of queues are maintained by the Mac operating system; the example won't list all of them, but some of the more important ones. Queues are linked lists of information blocks. Each block contains the address of the next list element in its first four bytes, a 16-bit number indicating the queue type in its next 2 bytes, and variable length information thereafter, as indicated below:

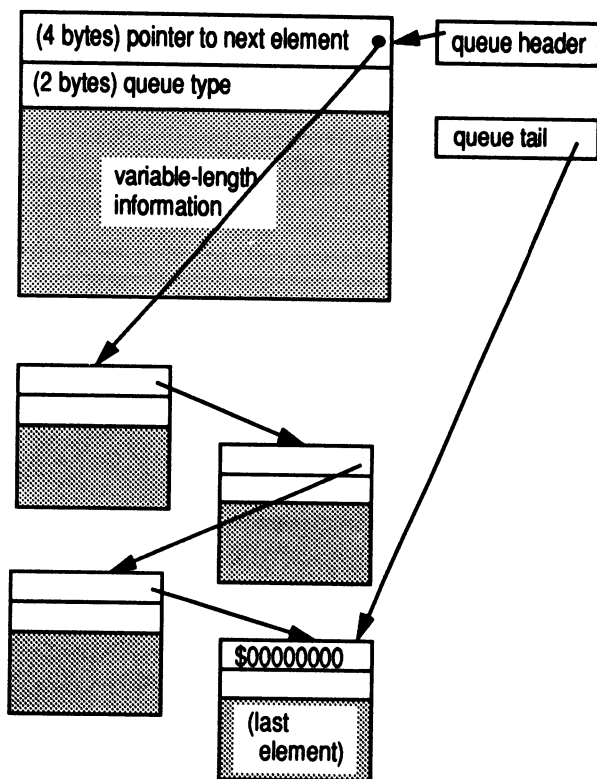


Fig.1: general structure of a Macintosh queue

The first element of a queue is called the *queue header* and the last one the *queue tail*. The queue tail has a link address of zero, meaning no more elements will follow. One can access the elements of a queue through a pointer to its header which is

usually kept in a low memory global variable. For some of the queues the tail's location is kept in another global variable for faster access.

The low memory addresses of the queues that our utility will display - vertical blanking queue, volume control block queue, drive queue, file system queue, event queue, deferred task queue and time manager queue - are defined at the beginning of the listing. (Since the address of the TM queue is not given in IM, I dug it out of a disassembly of the *_InsTime* trap.)

Structure of the program

We'll set up a Mach 2 terminal task that contains the usual Apple, File and Edit menus (for desk accessories). The task window will contain a bar display for each queue which gives a visual indication of the size of the queue. The display bars are 'click-sensitive'; when the mouse is clicked on one of them, a popup menu is displayed which shows the name of the queue. When this name is selected, a second window will be opened with information about the queue elements. That second window will disappear on a mouse click.

Inter-task communication in Mach2

I was happily hacking away on the example, got the rectangles drawn in the right places, the popup menus setup (note the slight difference in the interface that's used here - built-in - from my code in MT V4#4), when I got a row of *bad* bombs trying to write text into a second window from the queue-monitoring task. Just ADDing a new window, then doing a *_SetPort* and using the Mach2 text I/O routines wouldn't work at all. When it finally dawned on me... what was I actually doing? The main task detects a mouse down in the content region of its window and calls the content handler (see listing) to find out which rectangle it has been clicked in, which menu to display, and to output the text in the second window. Now that is a totally trivial piece of code in a single task program like one would write in C or Pascal most of the time. In Mach2, things are different.

The mouse down event is not detected by the task that is supposed to handle it, but by the (in)famous I-O task, present in any Mach2 application. That task calls *WaitNextEvent*, detects which window the mouse was clicked in, and calls the event handler of the task that owns the window. So while we're executing the event handler, we're actually in the context of the I-O task! The standard Mach2 console output would be sent to the window associated with the I-O task. Unfortunately, there is no such window - therefore bomb.

The bottom line is that you can't use the Forth text I/O routines from event handlers that are called through the I-O task.

You have to rely on the means that Mach2 provides to do inter-task communication and make a terminal task do the text output.

Mach2 text output from event handlers

How is this achieved? The User area of a Mach2 task contains two locations, *UserData* and *UserVector*. When a non-zero value is stored in *UserData*, the routine pointed to by *UserVector* will be executed. So all we need to do is define a user vector handling routine, and have the content handler of the task store some message in *UserData*.

The content handler of our main window simply stores the menu ID of the selected popup menu in *UserData*. The user vector handler finds this menu ID (it gets passed on the stack), and calls the corresponding queue display routine.

The queue display routines use a second window, *qInfo*. In order to have the output appear in that window and not on top of our bar display, we have to change the task window pointer momentarily; this is done by storing the *qInfo* window pointer in the user variable *taskWindowPointer*, doing the output and later restoring the original pointer.

Queues displayed

The utility displays seven queues (and you can easily add more): vertical blanking, volume control block, drive, event, file system, deferred task, and time manager queues, in that order. The display bars are not labeled, but the popup menu will show you the name of the queue when you click on a bar.

When a popup menu has been selected, a queue list is displayed in a second window. The first two columns are the element number and queue type for each list element. The remaining columns vary according to the queue displayed. For the VBL queue, the VBL procedure pointer, the tick count and the 'phase' are listed. The VCB queue gives the name of each mounted volume, its drive number, driver and volume reference numbers, total number of allocation blocks, allocation block size and free blocks. The drive queue will give the drive number, driver reference number, file system ID, total number of blocks, and the status of the drive (locked disk, disk inserted, single or double sided for floppies). The status information is kept in the four bytes preceding the queue element (see IM II-128).

The event queue listing contains the *what*, *message*, *when*, *where*, and *modifiers* fields of the event. However, I'm sorry to say that in this case the listing window is next to useless, because the list is always empty when it is displayed. I'll let you think of a reason why. The bar display sometimes shows queued events, when the display window is brought to the front from a partially covered position (its activate and update events).

Since I never found any pending asynchronous file system requests, I could not test the display for the FS queue, so that display routine is empty, just like the deferred task display. The time manager queue list shows the procedure pointer and *count* field for each time manager task.

In my system (MacII, 80MB hard disk, 5 MB memory, Appletalk and file server installed) I found a total of 8 VBL tasks, 2 time manager tasks, 2 mounted volumes (hard disk and Ali-

sashare volume), and, interestingly, 5 different drives. I could make sense of three drives (SCSI disk, floppy drive and Alisashare disk), and can guess that a fourth 'drive', with the same file system ID as the Alisashare volume, might have something to do with the server. The fifth entry, with a queue type of 3, file system ID of \$43C, and driver reference number of zero, doesn't make sense to me. If someone has something to say about that point I'd be happier.

Why?

The crucial question - what is this utility good for? Besides satisfying the general curiosity of a system hacker, one important thing that I can imagine it would be helpful in is the detection of - you guessed it - viruses. At least background routines that install VBL or time manager tasks to do their malicious job might be spotted a lot easier when the queue display suddenly starts changing.

I did not add information about working directory control blocks, but you may do so if you like. The working directory queue header pointer is at \$372.

Mach2 news

Finally, Palo Alto Shipping has published a trap compiler that allows you to add new traps to the Mach2 system without writing assembly code. The code is freely accessible on the Mach2 GENie roundtable, but for those of you who don't have access to GENie, I'm including it on the source code disk together with some trap fixes for version 2.14. Sorry we can't print the complete code here because it is too long; also you'll have a lot of glue code typed in and debugged before you have finished typing the trap compiler in by hand. I'll nevertheless print a short extract of the description:

This utility allows you to define your own trap "glue." Thus as new system traps become available, you will be able to define their high-level interface symbolically (without using assembly language). It will also allow you to define a substitute syntax to the CALL interface, in case you need to modify (fix?) one of the existing traps. Note that in the latter case, you do not actually change the existing CALL, you simply define a new syntax which can be used in place of the CALL sequence.

One might wonder whether it makes good sense to include a 2K utility in an application when all you need is a few trap calls (but, see my comments below). However, for casual use you might consider putting this code in your workspace, along with your commonly used constants, mach words, and other compiler utilities (that way they will always be available during your "experimental" sessions). If you are working on a serious application, I would suggest the following approach. Place all variables, mach words, and compiler utilities (such as this trap compiler) in a separate segment and when you finish your application use ResEdit to remove that very same segment from your finished application.

This will work because mach words, variables, and compiler utilities (such as this trap compiler) do NOT need to be in memory during the execution of the code which they produce. Generally

speaking, any word which only produces in-line code, most immediate words, or any child word which does not reference its parent at run-time can safely be removed from a finished application. If you are really concerned about making the smallest possible applications, then this is a technique which you should always use (as a final step, when you are completely finished with your program). If it seems risky to remove a segment from a finished application, just remember that there is NO WAY you can run code in another segment if that segment has NO jump-table entries (i.e. if you can't get to the code, why include it in your application?)

Mach words, variables, and (most) compiler words don't use or create jump-table entries. This same principle is why you never have to mark mach words or variables as GLOBAL (there is one exception, if you write a CODE word which explicitly states "JSR <mach word>" or "LEA <mach word>" then the defined instance of that <mach word> must be in memory at run-time. Thus if <mach word> exists in another segment, it will need to be marked GLOBAL (however, it's always ok to say LEA <variable> or MOVE.L <variable>, thus variables "never" need to be marked as GLOBAL).

Waymen @ PASC

Thanks for that utility and the comments on segmentation.
Till next month.

Listing 1: System queue display

```
\ mini task to display system queues
\ © 1988 J. Langowski / MacTutor
```

```
only forth also assembler also mac
```

```
$160 constant vblqhdr
$356 constant vcbqhdr
$308 constant drvqhdr
$14a constant evtqhdr
$360 constant fsqhdr
$d92 constant dtqhdr
$b30 constant timevars
$11c constant utablebase
```

```
72 user taskwindowpointer
92 user (type)
108 user taskmenubar
144 user uservector
148 user userdata
152 user content-hook
164 user goaway-hook
168 user update-hook
172 user activate-hook
```

```
2 CONSTANT Message
10 CONSTANT Where
14 CONSTANT Modifiers
1 CONSTANT ActivateMask
```

```
300 constant appleid
301 constant fileid
302 constant editid
```

```
152 CONSTANT WrefCon
```

```
create rect1 20 w, 20 w, 90 w, 28 w,
create rect2 20 w, 35 w, 90 w, 43 w,
```

```
create rect3 20 w, 50 w, 90 w, 58 w,
create rect4 20 w, 65 w, 90 w, 73 w,
create rect5 20 w, 80 w, 90 w, 88 w,
create rect6 20 w, 95 w, 90 w, 103 w,
create rect7 20 w, 110 w, 90 w, 118 w,
```

```
CREATE APPLESTRING 01 C, $14 C,
```

```
NEW.MBAR queueBar
```

```
NEW.MENU AppleMenu
APPLESTRING AppleMenu TITLE
0 APPLEID AppleMenu BOUNDS
" About Queues ...;(-" AppleMenu ITEMS
```

```
NEW.MENU FileMenu
" File" FileMenu TITLE
0 FileID FileMenu BOUNDS
" Quit" FileMenu ITEMS
```

```
NEW.MENU EditMenu
" Edit" EditMenu TITLE
0 EDITID EditMenu BOUNDS
" (Undo/Z;(-;Cut/K;Copy/C;Paste/V;Clear" EditMenu ITEMS
```

```
NEW.MENU vblmenu
" vbl" vblmenu TITLE
-1 150 vblmenu BOUNDS
" VBL Tasks;(-" vblmenu ITEMS
```

```
NEW.MENU vcbmenu
" vcb" vcbmenu TITLE
-1 151 vcbmenu BOUNDS
" Vol contrl blks;(-" vcbmenu ITEMS
```

```
NEW.MENU drvmenu
" drv" drvmenu TITLE
-1 152 drvmenu BOUNDS
" Drives;(-" drvmenu ITEMS
```

```
NEW.MENU evtmenu
" evt" evtmenu TITLE
-1 153 evtmenu BOUNDS
" Events [???];(-" evtmenu ITEMS
```

```
NEW.MENU fsmenu
" fs" fsmenu TITLE
-1 154 fsmenu BOUNDS
" File System;(-" fsmenu ITEMS
```

```
NEW.MENU dtmenu
" dt" dtmenu TITLE
-1 155 dtmenu BOUNDS
" Def Tasks;(-" dtmenu ITEMS
```

```
NEW.MENU tmmenu
" tm" tmmenu TITLE
-1 156 tmmenu BOUNDS
" Time manager;(-" tmmenu ITEMS
```

```
NEW.WINDOW SysQueue
" System Queues" SysQueue TITLE
250 50 350 250 SysQueue BOUNDS
Rounded Visible CloseBox GrowBox SysQueue ITEMS
```

```
NEW.WINDOW qInfo
" Queue Info" qInfo TITLE
50 50 250 500 qInfo BOUNDS
NoGrow Invisible NoCloseBox NoGrowBox qInfo ITEMS
```

```
500 2000 terminal queues
```

```
CODE unpack
MOVE.L (A6),D0
```

```

CLR.L D1
MOVE.W D0,D1
CLR.W D0
SWAP.W D0
MOVE.L D0,(A6)
MOVE.L D1,-(A6)
RTS
END-CODE MACH

: beep ( n )
  0 do 1 call sysbeep loop
;

: wait ( #ticks | time - )
  call tickcount #ticks +> time
  begin pause
    call tickcount time >
  until
;

: popup.select ( menu pt | point - menuID item# )
  pt -> point
  ^ point call localtoGlobal
  menu @ point unpack 1
  call popupmenuselect
  unpack
;

: do.content ( | pt - )
  CALL FrontWindow CALL SetPort
  qInfo call HideWindow

RUN-CONTENT
EVENT-RECORD Where + @ -> pt
^ pt CALL GlobalToLocal

0
pt rect1 CALL PtInRect
  IF drop vblmenu THEN
pt rect2 CALL PtInRect
  IF drop vcbmenu THEN
pt rect3 CALL PtInRect
  IF drop drvmenu THEN
pt rect4 CALL PtInRect
  IF drop evtmenu THEN
pt rect5 CALL PtInRect
  IF drop fsmenu THEN
pt rect6 CALL PtInRect
  IF drop dtmenu THEN
pt rect7 CALL PtInRect
  IF drop tmmenu THEN

?dup IF ( rectangle was selected )
  pt popup.select

  IF( popup selection was made )
    ( menuID ) userData task-> queues !
  THEN
THEN

: draw.rects
  rect1 call framerect
  rect2 call framerect
  rect3 call framerect
  rect4 call framerect
  rect5 call framerect
  rect6 call framerect
  rect7 call framerect
;

: clr.rects
  rect1 call eraserect
  rect2 call eraserect

```

```

rect3 call eraserect
rect4 call eraserect
rect5 call eraserect
rect6 call eraserect
rect7 call eraserect
;

: blackBar ( rect pixels | locBR locTL )
  rect ^ locTL 8 cmove
  ^ locBR w@ ( bottom ) pixels -
  ^ locTL w!
  ^ locTL call paintrect
;

: #elems ( qhdr | elems - #.of.queue.elements )
  0 -> elems
  2 +> qhdr
  begin
    qhdr @ ?dup while
      -> qhdr
      1 +> elems
  repeat
  elems
;

: display.queues ( | - )
  clr.rects
  draw.rects
  rect1 vblqhdr #elems 4* blackBar
  rect2 vcbqhdr #elems 4* blackBar
  rect3 drvqhdr #elems 4* blackBar
  rect4 evtqhdr #elems 4* blackBar
  rect5 fsqhadr #elems 4* blackBar
  rect6 dtqhadr #elems 4* blackBar
  rect7 timevars @ 8 + #elems 4* blackBar
;

: dsp.vbl ( | qelemPtr n - )
  cls
  qinfo " Vertical Blanking Tasks"
  call SetWTitle
  " _____" cr
  " task# qtype ProcPtr Count Phase" cr
  " _____" cr
  vblqhdr 2+ -> qelemPtr
  1 -> n
  begin
    qelemPtr @ ?dup while
      -> qelemPtr
      n 3 .r 3 spaces
      hex
      qelemPtr 4 + w@ 5 .r space
      qelemPtr 6 + @ 8 .r space
      qelemPtr 10 + w@ 5 .r space
      qelemPtr 12 + w@ 5 .r cr
      decimal
      1 +> n
  repeat
;

: dsp.vcb ( | qelemPtr n - )
  cls
  qinfo " Volume Control Blocks"
  call SetWTitle
  " _____" cr
  " vcb# qtype Volume name Drive dRef# vRef# #blks blksz" cr
  " free" cr
  " _____" cr
  " _____" cr
  vcbqhdr 2+ -> qelemPtr
  1 -> n
  begin
    qelemPtr @ ?dup while

```

```

-> qelemptr
n 3 .r 3 spaces
hex
qelemptr 4 + w@ 5 .r space
qelemptr 44 + count dup rot swap type
27 swap - spaces
qelemptr 72 + w@ 5 .r space
qelemptr 74 + w@ 5 .r space
qelemptr 78 + w@ 5 .r space
qelemptr 26 + w@ 5 .r space
qelemptr 28 + @ 5 .r space
qelemptr 42 + w@ 4 .r cr
decimal
1 +> n
repeat
;

: dsp.driv ( | qelemPtr n - )
cls
qinfo " Drives"
call SetWTitle
" _____" cr
" drv# qtype Drive dRef# FSID #blks l dd ss" cr
" _____" cr
drvhqdr 2+ -> qelemptr
1 -> n
begin
qelemptr @ ?dup while
-> qelemptr
n 3 .r 3 spaces
hex
qelemptr 4 + w@ 5 .r space
qelemptr 6 + w@ 5 .r space
qelemptr 8 + w@ 5 .r space
qelemptr 10 + w@ 4 .r space
qelemptr 12 + w@
qelemptr 4 + w@ 1 = IF
qelemptr 14 + w@ 65536 * +
THEN
8 .r space
qelemptr 4- c@ $80 AND
IF ascii y emit ELSE ascii n emit THEN space
qelemptr 3- c@ 2 .r 2 spaces
qelemptr 1- c@ $80 AND
IF ascii n emit ELSE ascii y emit THEN cr
decimal
1 +> n
repeat
;

: dsp.evt ( | qelemPtr n - )
cls
qinfo " Queued Events"
call SetWTitle
" _____" cr
" drv# qtype What Message When Where Mods" cr
" _____" cr
evtqhdr 2+ -> qelemptr
1 -> n
begin
qelemptr @ ?dup while
-> qelemptr
n 3 .r 3 spaces
hex
qelemptr 4 + w@ 5 .r space
qelemptr 6 + w@ 5 .r space
qelemptr 8 + @ 8 .r space
qelemptr 12 + @ 8 .r space
qelemptr 16 + @ 8 .r space
qelemptr 20 + w@ 4 .r cr
decimal
1 +> n
repeat
;

```

```

: dsp.fs
cls
qinfo " File System Requests"
call SetWTitle
;

: dsp.dt
cls
qinfo " Deferred Tasks"
call SetWTitle
;

: dsp.tm ( | qelemPtr n - )
cls
qinfo " Time manager"
call SetWTitle
" _____" cr
" task# qtype ProcPtr Count" cr
" _____" cr
timeVars @ 10 + -> qelemptr
1 -> n
begin
qelemptr @ ?dup while
-> qelemptr
n 3 .r 3 spaces
hex
qelemptr 4 + w@ 5 .r space
qelemptr 6 + @ 8 .r space
qelemptr 10 + w@ 5 .r cr
decimal
1 +> n
repeat
;

: do.user
qInfo dup call showwindow call selectwindow
qInfo taskwindowpointer !
( menuID ) CASE
150 OF dsp.vb1 ENDOF
151 OF dsp.vcb ENDOF
152 OF dsp.driv ENDOF
153 OF dsp.evt ENDOF
154 OF dsp.fs ENDOF
155 OF dsp.dt ENDOF
156 OF dsp.tm ENDOF
ENDCASE
SysQueue taskwindowpointer !
;

: do.update ( | pt - )
sysQueue call setport
draw.rects
run-update
;

: do.activate
event-record modifiers + w@
1 AND \ Activate event?
IF
call DrawMenuBar
ELSE
THEN
;

: do.close
bye
;

: INIT-MBAR
queueBar ADD
queueBar APPLEMENU ADD
APPLEMENU @ ascii DRVR CALL ADDRESSMENU
queueBar FileMenu ADD

```

```

queueBar EditMenu ADD
queueBar vblmenu add
queueBar vcbmenu add
queueBar drvmenu add
queueBar evtmenu add
queueBar fsmenu add
queueBar dtmenu add
queueBar tmmenu add
;

: DO-APPLE ( item# | [ 32 lallot ] daName - )
  item# 1 =
  IF3 beep

  ELSE AppleMenu @
    item# ~ DAName CALL GetItem
    ~ DAName CALL OpenDeskAcc DROP
  THEN
;

: do-file
  drop bye
;

: MBAR-HANDLER ( item# menuID - )
  CASE
  APPLEID OF DO-APPLE ENDOF
  FILEID OF DO-FILE ENDOF
  drop
  ENDCASE
  @ CALL HILITEMENU

```

```

;

: go.queue ( | mb - )
  activate

  ['] do.content content-hook !
  ['] do.update update-hook !
  ['] mbar-handler menu-vector !
  ['] do.activate activate-hook !
  ['] do.close goaway-hook !
  ['] do.user uservector !

  begin
    pause
    sysQueue call setport
    display.queues
    60 wait
  again
;

: start
  SysQueue add
  QInfo add
  SysQueue queues build
  SysQueue WRefCon + @
  QInfo WRefCon + !
  SysQueue dup call selectwindow call setport
  init-mbar
  queueBar queues mbar>task
  queues go.queue
;

```



Forth Forum

Record Definitions



Jörg Langowski
MacTutor Editorial Staff
Volume 5, Number 2

"Record definitions in Mach2"

Record structures and arrays are not part of standard Forth implementations. More than two years ago, in V2#7, I had given an example how to implement records. Mach2 has evolved since then, and so have ways of implementing new data structures, as you can see in the Object Forth project by Wayne Joerding that we recently discussed. For those of you who do not want a full object-oriented system, but still ways of defining data structures in an easy way, I have found two examples on the GENie bulletin boards. Those examples show two fundamentally different approaches to deal with record definitions.

'Local' field names - method 1

The problem in setting up the Forth compiler to deal with record definition in a proper way is somewhat similar to implementing an object-oriented programming system. That is, just like a message is local to an object, and the same message may cause different effects on different objects, a field name should be local to a record. In the Pascal record definitions

```
rec1 = record
    x: real;
    i: integer;
    y: real;
end;

rec2 = record
    y: real;
    j: integer;
    x: real;
end;
```

the field *x* would create a different offset into a record of type *rec2* than for a *rec1* type; and *rec1.i*, *rec2.j* would be valid while *rec1.j*, *rec2.i* would not. So if we define a field name as some kind of Forth word, this word should be in some 'local vocabulary' that belongs to the record definition and is only visible while the field reference is resolved.

The other requirement is that we should be able to pass a record as a parameter to a routine, so that given the pointer to a record on the stack, a Forth definition would know how to resolve the field reference. In a strongly typed language like Pascal this is easy; field references into record formal parameters can be resolved at compile time because the procedure arguments are of defined type. In Forth, typically the address of a data structure would be passed on the stack. However, at compile time there is no way we can restrict the type of argument that this address might later point to at run time! This problem could only be solved by type checking built into the record definition and deferring the resolution of the field reference to run time, some sort of 'late binding'.

The first method of record definition (Listing 1), written by Waymen Askey of Palo Alto Shipping (I added some minor modifications, like floating point and array support), creates a local dictionary for each record template in the Forth dictionary space. When a record template is defined, using the syntax

```
template rec1
    :real x
    :word i
    :byte c
tend
```

its field names *x*, *i* and *c* are compiled into the dictionary together with relevant information for resolving the references. At the end of the template declaration, the dictionary links are changed in such a way that the 'local' names are skipped when the dictionary is searched. Let's declare a record:

```
rec1 structure myRec
```

A field of this record is later accessed by using the structure fetch/store words, *s@* and *s!*.

myRec x s@ will put the value in field *x* of *myrec* on the floating point stack, and *myRec i s@* will put the word value of field *i* on the stack. The trick Waymen used was to build some intelligence into the fetch/store words. When the record and field words, *myRec* and *x* for example, are executed or compiled into a definition, field type and offset are determined and kept in global variables. The *s@* word will check these variables and know how to access the field, whether - in immediate execution - to do a byte, word or long word fetch, addressing into an array, or a ten-byte fetch onto the floating point stack for a real number; or at compile time create code that will do these things later.

The drawback of this approach is that field references can only be resolved at compile or immediate execution time. If we wanted to write a word that operates on a record whose address is passed on the stack, we couldn't use the field names that were defined in the record template - they are only valid right after a record name was executed or compiled. Therefore, a definition like

```
: getX ( myRec - ) myRec x s@ ;
```

must fail because *myRec* is a local variable, not a record name.

An example how to use this method of record declaration with various field types is given at the end of the listing. You see the drawback: Even though the record fields *wavelength*, *temperature*, and *angle* are all themselves structures of the same type *parameter*, there is no way to factor out the common code in


```

cr curve1 wavelength name s^ count type ." = "
  curve1 wavelength value s@ f.
  curve1 wavelength unit s^ count type
cr curve1 temperature name s^ count type ." = "
  curve1 temperature value s@ f.
  curve1 temperature unit s^ count type
cr curve1 angle name s^ count type ." = "
  curve1 angle value s@ f.
  curve1 angle unit s^ count type

```

by using a word that would just print name, value and unit of any given parameter. If this problem was resolved, the record compiler would almost be perfect.

'Global' field names - method 2

Listing 2 shows a much simpler approach to structure definitions that does not do type checking. I downloaded this code from the Forth Roundtable on GENie, and unfortunately have not the slightest idea who the author is. All I could find out was that the original code was probably posted on the East Coast Forth Board.

However, since this code solves one of our problems, record passing as formal parameters, I'd like to print it here. Its strategy is much more like that of the structure words built into MacForth Plus. Here, a record template is defined like

```

RECORD Rectangle
  Global SHORT: Top
  Global Short: Left
  Global Short: Bottom
  Global SHORT: Right
END.RECORD

Variable myRect Rectangle 4 - VALL0T ;

```

so the record name, when executed, simply leaves the record length on the stack for later ALLOT or VALL0T. The field names are words which add the field offset to an existing address on the stack, so they can be used in any context. We have to check ourselves whether the address is a valid record address and whether the field referenced actually exists in that record (if we care at all). All field names are global, and therefore must be unique; no two different record declarations can have fields of the same name at different offsets.

This approach is not so different from the very basic one that I used in most of my examples, where I simply defined field names as constants and added the offset to the record address.

What the Macintosh Forth world needs is really a combination of the two approaches, with type checking at compile time and local field names for convenience, and a possibility to resolve field references on record *addresses* at compile time without too much overhead. If one knew the type of the record passed on the stack ahead of time (which is usually the case), one could probably define some 'field reference resolution word' which computes an offset given a template and a field name. I hope I can show you an example in one of my next columns.

Upcoming: an update to Wayne Joerding's Object Forth, and a review of PocketForth, a public domain 16-bit Forth that comes as an application *and* a desk accessory. Stay tuned.

Listing 1: Structure definitions with local field names

```

\ STRUCTURES 2.5 for the Macintosh MACH2
\ Jan 3, 1987 by Waymen Askey
\ edited, floating point & array addition by
\ J. Langowski @ MacTutor
\ This MACH2 extension is released for the public good;
\ however, for those planning commercial use of this code,
\ please notify me so that I might know of its intended use.
\
\      Waymen Askey @ PASC
\
\ also GENie MACH2 RoundTable.

```

```

only mac also sane also forth definitions
( VARIABLES used in STRUCTURE 2.5 )
decimal
variable current.template
variable op.type
variable A5offset ( holds the A5 offset to a structure )

```

```

( CODE word utilities used in STRUCTURE 2.5 )
code var.link ( - a | variable link pointer )
  lea $F7F8(A5),A0
  move.l A0,-(A6)
  rts
end-code

```

```

code a5@ ( - a )
  move.l A5,-(A6)
  rts
end-code mach

```

```

code get.field ( a1 a2 - a3 -1 or 0 | searches templates )
  ( a1=template, a2= pad, a3=field pointer, 0 if not found )
  move.l (A6)+,D2
  move.l (A6)+,D3
  moveq.l #0,D1
  moveq.l #0,D0
  @start movea.l D3,A1
  movea.l D2,A0
  move.b (A1)+,D1 ( link to next field )
  beq.s @end ( if link=0, field not found )
  move.b (A1),D0
  @loop cmpm.b (A1)+,(A0)+
  dbne D0,@loop
  beq.s @found
  add.l D1,D3 ( increment field pointer )
  bra.s @start
  @found movea.l D3,A1
  move.b 1(A1),D1 ( get string count )
  addq.w #2,D1
  btst #0,D1 ( test for odd count )
  beq.s @even
  addq.w #1,D1
  @even add.l D1,D3
  moveq.l #-1,D1
  move.l D3,-(A6)
  @end move.l D1,-(A6)
  rts
end-code

```

```

code >sr ( n - | push value onto subroutine stack )
  move.l (A6)+,-(A7)
  rts
end-code mach

```

```

code sr> ( - n | pop value from subroutine stack )
  move.l (A7)+,-(A6)
  rts
end-code mach

```

```

code sr@ ( - n | copy value from subroutine stack )
  move.l (A7),-(A6)
  rts
end-code mach

```

```

( Miscellaneous utility words used in STRUCTURE 2.5 )
: >even ( a - a' |
    word aligns address, i.e. rounds up to even)
    dup 1 and + ;

: >odd ( a - a' | odd aligns address, rounds up to odd )
    1 or ;

: needed ( n - | checks for at least n items on stack )
    depth 1- > abort" Missing needed stack item(s)! " ;

( Brute-force machine code words )
: ncode,
( n1...n - | machine code defining word, stuffs n words )
    create dup needed dup 2* w,
    0 do w, loop
does> ( - | compiles machine code )
    dup 2+ swap dup w@ +
    do i w@ w, -2 +loop ;

hex
( define some machine code "stuff" words )
41ED 1 ncode, lea_d(a5),a0
4EBA 1 ncode, jsr_d(PC)
4EAD 1 ncode, jsr_d(A5)
( LEA and JSR also need a word of extension for displacement )
2D3C 1 ncode, move.l_#,-(A6)
    ( plus a long extension for # )
2D08 1 ncode, move.l_a0,-(a6)
4E75 1 ncode, rts,
( The following expect an address to be in A0 )
7000 1010 2D00 3 ncode, byte@
7000 3010 2D00 3 ncode, word@
2D10 1 ncode, long@
201E 1080 2 ncode, byte!
201E 3080 2 ncode, word!
209E 1 ncode, long!
\ disassemble the following to check how they work.
\ Exercise for the reader... - JL
5187 5587 2247 22d8 22d8 32d8 6 ncode, real@
2247 20d9 20d9 30d9 5087 5487 6 ncode, real!
201e e580 2d30 0000 4 ncode, array@
201e e580 219e 0000 4 ncode, array!
201e e380 4281 3230 0000 2d01 6 ncode, warray@
201e e380 221e 3181 0000 5 ncode, warray!
decimal

( Dictionary header, name, and struct link words )
: link>name ( lfa - 'nf | 'nf points to header length byte)
    4 + ;

: name.count
    ( 'nf - 'nf+1 n | dictionary header name count)
    count 31 and ;

: link>segment
    ( lfa - 'sf | 'sf is the dictionary segment field address)
    link>name name.count + >even ;

: link>parameter
    ( lfa - 'pf | 'pf is the parameter field pointer)
    link>segment 2+ ;

: link>struct ( lfa - struct.fields )
    link>segment 4 + ;

: jsr_d(PC), ( lfa - | compiles PC relative JSR)
    jsr_d(PC)
    link>body here - w, ;

: jsr_d(A5),
    ( lfa - | compiles A5 relative JSR, i.e. jump table )
    jsr_d(A5)
    link>parameter w@ w, ;

```

```

: struct.zero ( - lfa | returns lfa of struct.zero )
    " struct.zero" find drop ;

: nallot ( n - | allots n bytes in name space )
    np +! ;

: name, ( - parses and compiles text into name space.)
    32 word np @ over c@ 1+ dup >odd nallot cmove ;

: nc, ( n - | compiles byte into name space )
    np @ c! 1 nallot ;

: nw, ( n - | compiles word into name space )
    np @ w! 2 nallot ;

: n, ( n - | compiles long into name space )
    np @ ! 4 nallot ;

( TEMPLATE, STRUCTURE and field words )
: struct.error ( - )
    cr pad count type
    ." ? Error, unknown field or incomplete structure path! "
    abort ;

global
: template ( - here 0 | begins TEMPLATE definition )
    create here 0 2 allot
does> ( - template.size )
    dup w@ swap 4 - body>link current.template ! ;

: tend
    ( here n - | (T)emplate(END) ends template definition )
    swap w! 0 nw, ;

global
: afield ( size op.type - )
    create w, >even w,
does> ( here Toffset - here new.Toffset )
    ( Toffset means (T)emplate(OFFSET) )
    2dup 2+ w@ + >sr
    w@ np @ >sr 1 nallot name,
    0 nc, ( field type=0 ) nc, ( op.type )
    nw, ( Toffset ) np @ sr@ - sr> c! ( field link )
    sr> ;

( The following op.types are reserved and defined below )
( 06 byte, 12 word, 18 long, 24 string,
    30 real, 36 struct, 42 array, 48 warray )

( size.in.bytes op.type AFIELD named.afield.type )
1 06 afield :byte
2 12 afield :word
4 18 afield :long
10 30 afield :real

: :string ( here Toffset size - here Toffset+size+1 )
    3 needed 1+ over + >even swap
    np @ >sr 1 nallot name,
    0 nc, ( field type=0 ) 24 nc, ( op.type=24)
    nw, ( Toffset ) np @ sr@ - sr> c! ( field link ) ;

: :array ( here Toffset size - here Toffset+size+1 )
    3 needed 4* over + swap np @ >sr 1 nallot name,
    0 nc, ( field type=0 ) 42 nc, ( op.type=42)
    nw, ( Toffset ) np @ sr@ - sr> c! ( field link ) ;

: :warray ( here Toffset size - here Toffset+size+1 )
    3 needed 2* over + swap np @ >sr 1 nallot name,
    0 nc, ( field type=0 ) 48 nc, ( op.type=48)
    nw, ( Toffset ) np @ sr@ - sr> c! ( field link ) ;

: :struct ( here Toffset size - here Toffset+size )
    3 needed over + >even swap

```

```

    np @ >sr 1 hallot name,
    06 nc, ( field type=06 ) 36 nc, ( op.type=36 )
        nw, ( Toffset )
current.template @ struct.zero - n, ( template link )
np @ sr@ - sr> c! ( field link );

: >pad ( a - | moves string to pad )
pad over c@ 1+ cmove ;

: make.var.link ( | name.pointer var.pointer vlink - )
np @ -> name.pointer var.link @ -> var.pointer
name.pointer var.link !
name.pointer var.pointer - -> vlink
name.pointer dup 1 and + -> name.pointer
vlink name.pointer !
name.pointer 4 + np ! ;

( Decision table for field type decode )
: do.afield ( ^field.type - true )
1+ dup c@ op.type ! 1+ w@ A5offset +! -1 ;

: do.bfield ( ^field.type - new.template false )
dup 1+ dup c@ op.type ! 1+ w@ A5offset +!
4 + @ struct.zero + link>struct 0 ;

: rts rts, ; immediate

( D0.FIELD table entries decode field data )
( afield's are simple :BYTE, :WORD,
:LONG, :STRING types )
( bfield's are :STRUCT fields )

create do.field ( field_type table_offset/type )
] do.afield rts ( afield 0 )
do.bfield rts ( bfield 6 )
[ ( end of current table )

global
: make.struct ( template.link A5offset - )
( This is the word which must resolve a structure reference. )
A5offset ! ( A5 displacement for the struct )
36 op.type ! ( set default op.type to struct )
struct.zero + link>struct ( template.address - )
begin
32 word >pad
pad get.field
if ( field found )
dup c@ do.field + execute
else ( field not found )
pad find 1 =
if
link>body execute -1
else
struct.error
then
then
until ;

hex
: structure
( n - | creates structure allotting n bytes in variable space )
1 needed create immediate make.var.link
-4 allot lea_d(a5),a0 vp @ w, ( variable-like beginning )
move.1_*,-(A6) current.template @ struct.zero - ,
move.1_*,-(A6) vp @ ,
" make.struct" find drop dup link>segment w@ 0=
if jsr_d(PC), else jsr_d(A5), then
rts,
vallot ;
decimal

( STRUCTURE operators )
: compileA5 ( - | compiles A5 reference )
lea_d(a5),a0 a5offset @ w, ;

```

```

: pushA5 ( - | executes A5 var reference )
a5offset @ a5@ + ;

: do.bit ( - ) ( I'm lazy, define your own. W. Askey )
cr ." BIT operations are yet undefined!" abort ;

: do.struct ( - ) ( Fetch/store doesn't make sense here. )
cr ." STRUCTURE fetch/store operations are undefined!"
abort ;

: do.string ( - ) ( If you wish, define your own. )
cr ." STRING fetch/store operations are undefined!" abort ;

: do.byte@ ( f - )
if compileA5 byte@ else pushA5 c@ then ;

: do.word@ ( f - )
if compileA5 word@ else pushA5 w@ then ;

: do.long@ ( f - )
if compileA5 long@ else pushA5 @ then ;

: do.array@ ( idx f - )
if compileA5 array@ else 4* pushA5 + @ then ;

: do.warray@ ( idx f - )
if compileA5 warray@ else 2* pushA5 + w@ then ;

: do.real@ ( f - )
if compileA5 real@ else pushA5 f@ then ;

( Decision table for fetch )
create op.table@ ( op.types are offsets into this table )
] do.bit rts ( op.type = 0 )
do.byte@ rts ( " " = 6 )
do.word@ rts ( " " = 12 )
do.long@ rts ( " " = 18 etc, etc. )
do.string rts
do.real@ rts
do.struct rts
do.array@ rts
do.warray@ rts
[

: do.byte! ( f - )
if compileA5 byte! else pushA5 c! then ;

: do.word! ( f - )
if compileA5 word! else pushA5 w! then ;

: do.long! ( f - )
if compileA5 long! else pushA5 ! then ;

: do.array! ( idx f - )
if compileA5 array! else 4* pushA5 + ! then ;

: do.warray! ( idx f - )
if compileA5 warray! else 2* pushA5 + w! then ;

: do.real! ( f - )
if compileA5 real! else pushA5 f! then ;

create op.table! ( decision table for store )
] do.bit rts
do.byte! rts
do.word! rts
do.long! rts
do.string rts
do.real! rts
do.struct rts
do.array! rts
do.warray! rts
[

```

```

: s^ ( - a | returns pointer to structure field )
( ALL field types are allowed. i.e. strings, struct, etc. )
state @
if compileA5 move.l a0, -(a6) else pushA5 then
; immediate

: s@ ( - data | Fetch field contents, data type smart )
state @
op.type @ op.table@ + execute ; immediate

: s! ( data - | Store into field, data type smart )
state @
op.type @ op.table! + execute ; immediate

: stype ( - op.type | returns the op.type of a field )
op.type @ state @
if [compile] literal then
; immediate
( Examples of structure usage. Data Storage is limited to the
approximately 32K global area referenced off of register A5 -
just as for regular MACH2 variables. Structure references have
a REQUIRED syntax, it is best NOT to use any non-STRUCTURE
Forth words when between field names in a structure calling
sequence. That is, please end each structure reference prior
to any DUP's, SWAP's, etc. The structure pointer operator - S^
- may be used at any place in the structure calling sequence.
S^ will return the address of the field or structure itself.
Structures MUST be terminated with a defined structure
operator! The defined operators in this upload are S^, S@,
S!, and STYPE. WARNING, if you forget to terminate a struc-
ture, no structure reference will be compiled and an error
message MAY NOT be given. Remember also that field names ARE
CASE SENSITIVE and LOCAL to the structure template. Last
comment, structures MAY be nested to any level. )

```

fp

```

template Point
:word x
:word y
tend

```

```

template Rect
:word top
:word left
:word bottom
:word right
tend ( TEND ends template definition )

```

```

\ example for FP parameters
template parameter
30:string name
:real value
30:string unit
tend

```

```

template measurement
:long date \ in internal Mac format
80:string title
255:string descriptor
parameter :struct wavelength
parameter :struct temperature
parameter :struct angle
256:array time
256:array counts
tend

```

measurement structure curve1

```

: testarray
100 0 do i 4* i curve1 time s! loop
100 0 do i curve1 time s@ . cr loop;

```

```

: .date ( DateTime DateForm ) ( | [ 40 1allot ] mydate - )
8 shift ^ mydate call IUDateString ^ mydate count type;

: read.int
begin
pad 1+ 80 expect span @ pad c! pad number? not while
drop cr ." Illegal number [integer], reenter - "
repeat;

: read.float
begin
pad 1+ 80 expect span @ pad c! pad fnumber? not while
fdrop cr ." Illegal number [float], reenter - "
repeat;

: setup.curve1 ( | dattim - )
^ dattim call readdatetime drop @
cr ." Today is " 1 .date
cr ." Setting up parameters for curve 1."
dattim curve1 date s!
" lambda" dup c@ 1+ curve1 wavelength name s^ swap cmove
" T" dup c@ 1+ curve1 temperature name s^ swap cmove
" delta" dup c@ 1+ curve1 angle name s^ swap cmove
" [nm]" dup c@ 1+ curve1 wavelength unit s^ swap cmove
" [K]" dup c@ 1+ curve1 temperature unit s^ swap cmove
" [°]" dup c@ 1+ curve1 angle unit s^ swap cmove
cr ." Title (one line) - " cr pad 80 expect
span @ curve1 title s^ c!
pad curve1 title s^ 1+ span @ cmove
cr ." Description (one line) - " cr pad 80 expect
span @ curve1 descriptor s^ c!
pad curve1 descriptor s^ 1+ span @ cmove
cr ." lambda [nm] - " read.float curve1 wavelength value s!
cr ." T [K] - " read.float curve1 temperature value s!
cr ." delta [°] - " read.float curve1 angle value s!
\ example setup of 'measurement data'
20 0 do
i i curve1 time s!
i 100 * i curve1 counts s!
loop

cr ." End setup - " cr;

: dump.curve1 ( | [ 80 1allot ] mydate - )
cr ." Data taken on " curve1 date s@ 1 .date
cr curve1 title s^ count type
cr curve1 descriptor s^ count type
cr curve1 wavelength name s^ count type ." = "
curve1 wavelength value s@ f.
curve1 wavelength unit s^ count type
cr curve1 temperature name s^ count type ." = "
curve1 temperature value s@ f.
curve1 temperature unit s^ count type
cr curve1 angle name s^ count type ." = "
curve1 angle value s@ f.
curve1 angle unit s^ count type
cr ." data follows:"
20 0 do cr
i curve1 time s@ . space
i curve1 counts s@ .
loop
cr
;

```

Listing 2: Structure definitions from ECFB

\ downloaded from GENie J. L. Nov 1988
\ Originally from East Coast Forth Board,
\ author A. Nonymous
(This is a set of machforth routines for building records.
They allow you to build a named record with items of various
sizes. Executing the record name leaves the record size on the
stack, executing an item name leaves the offset of the item
into the record on the stack. It creates a template for the

record but not the actual record. Create the record with "create <name> <record name> allot" or "variable <name> <record name> 4 - valloT" depending if you want the entry in the dictionary or variable space)

VOCABULARY RECORDS (NEW VOCABULARY) ALSO RECORDS DEFINITIONS

```
Global
: Align ( n1 - [n1] or [n1 + 1] makes n word aligned )
    dup 2 mod + ; ( USED TO WORD ALIGN 2 & 4 BYTE ITEMS )
```

```
Global
: RECORD ( - a 0 )
    HERE 4 + CREATE 0 dup W, DOES> W0 + ;
    ( USED TO OPEN A RECORD )
```

```
Global
: BYTE: ( a n - a n1+1 )
    CREATE DUP W, 1+ DOES> W0 + ;
```

```
Global
: BYTES: ( a n1 n2 - a n1+n2 | AN ARRAY OF n2 bytes )
    CREATE OVER Align W, swap Align + DOES> W0 + ;
```

```
Global
: SHORT: ( a n1 - a n1+2 | 2 byte integer item )
    CREATE Align DUP W, 2+ DOES> W0 + ;
```

```
Global
: WORD: ( a n1 - a n1+2 | 2 byte integer item )
    CREATE Align DUP W, 2+ DOES> W0 + ;
```

```
Global
: BOOLEAN: ( a n1 - a n1+2 | 2 byte boolean item )
    CREATE Align DUP W, 2+ DOES> W0 + ;
```

```
Global
: SHORTS: ( a n1 n2 - a n1+n2*2 | an array of n2 shorts )
    CREATE OVER Align W, 2* Swap Align + DOES> W0 + ;
```

```
Global
: LONG: ( a n1 - a n1+4 | a 4 byte integer )
    CREATE Align DUP W, 4 + DOES> W0 + ;
```

```
Global
: POINTER: ( a n1 - a n1+4 | a 4 byte integer )
    CREATE Align DUP W, 4 + DOES> W0 + ;
```

```
Global
: LONGS:
    ( a n1 n2 - a n1+n2*4 | an array of n2 4 byte integers )
    CREATE OVER Align W, 4 * swap Align + DOES> W0 + ;
```

```
Global
: HANDLE: ( a n1 - a n1+4 | a handle, 4 byte, item )
    CREATE Align DUP W, 4 + DOES> W0 + ;
```

```
Global
```

```
: HANDLES: ( a n1 n2 - a n1+n2*4 | array of n2 handles )
    CREATE OVER Align W, 4 * swap Align + DOES> W0 + ;
```

```
Global
: ADDR: ( a n1 - a n1+4 | 4 byte address item, ie pointer )
    CREATE Align DUP W, 4 + DOES> W0 + ;
```

```
Global
: ADDRS: ( a n1 n2 - a n1+n2*4 | array of n2 addresses )
    CREATE OVER Align W, 4 * swap Align + DOES> W0 + ;
```

```
Global
: RECT: ( a n1 n2 - a n1+8 | a rect item )
    CREATE Align DUP W, 8 + DOES> W0 + ;
```

```
Global
: RECTS: ( a n1 n2 - a n1+n2*8 | an array of n2 rects )
    CREATE OVER Align W, 8 * swap Align + DOES> W0 + ;
```

```
Global
: STRING: ( a n1 n2 - a n1+n2+1 | a string item n2+1 long )
    CREATE OVER W, + 1+ DOES> W0 + ;
```

```
Global
: RECORD: ( a n1 n2 - a n1+n2 | a record item of size n2 )
    CREATE OVER Align W, swap Align + DOES> W0 + ;
```

```
Global
: END.RECORD
    ( Mainaddr size -|sets size of struct at a to n )
    Mainaddr W0 Size <
    IF Size MainAddr W! Then ;
    ( CLOSSES RECORD, STORES RECORD SIZE IN RECORD NAME)
```

```
Global
: SUB.REC ( - )
    Create 0 W, 2DUP Here 2- Rot Rot DOES> W0 + ;
    ( USE TO CREATE VARIANT RECORD ON THE END OF A RECORD)
```

```
Global
: END.SUB ( SubAddr MainAddr Size - )
    Size SubAddr W!
    MainAddr W0 Size <
    IF Size Align MainAddr W! Then ;
    ( USE TO CLOSE VARIANT RECORD )
```

ONLY MAC
ALSO FORTH
DEFINITIONS
ALSO RECORDS

```
Global
RECORD Rectangle
    Global SHORT: Top
    Global Short: Left
    Global Short: Bottom
    Global SHORT: Right
```

```
End.RECORD
```

```
Global
: rect Variable Rectangle 4 - VALLOT ;
( CREATES A RECTANGLE RECORD IN THE VARIABLE SPACE )
```



Forth Forum

System CDEV



Jörg Langowski
MacTutor Editorial Staff
Volume 5, Number 5

"System configuration cdev"

There is a nice public domain CDEV utility, called Systat, that shows the system configuration you're running on: machine type, CPU, floating point hardware, etc. Ever since my last system update, this CDEV doesn't run anymore on my Mac II. Don't know what I did wrong or what INIT I installed that conflicts with it. So to have that utility back, I decided to write my own - in Mach2 - to give you an example how to write CDEVs and how to use the SysEnvironments trap [which is unfortunately left unimplemented in Mach 2.14].

SysEnvironments

The Macintosh world has evolved a long way from the one-configuration era of the 128K Mac. There are now - at least - 8 different types of machines on which an application could run, the 512K old ROM Mac (yes, some still have one), the Lisa, the 512KE, Mac Plus, Mac SE, SE/030, Mac II and Mac Ix. I'm not counting the 'small' MacII with fewer slots since I haven't seen that machine as of this writing.

Add to this variety of machines all the different accessories that could be present, such as large screens or accelerator cards, and you end up with an impressive array of different configurations. The confusion is far from being as bad as in the MS-DOS world, but it's there; no wonder that some applications start having problems running on all possible configurations.

It is for this reason that the SysEnvironments trap - described in IM V-6 - has been added to the system. This routine returns information about the system configuration the program is running on, so that you can exit gracefully from your application if - for instance - color Quickdraw is needed but not available.

SysEnvironments accepts two arguments: a pointer to a data structure called a *sysEnvRec* in A0 and a version number in D0. The version number is used to maintain upward compatibility. Right now, two versions of the trap call exist, version 1 and 2, which both return the same information. If, at a later time, new versions are defined that return more or other information, the old calls can still be used, returning the data in the same old format. SysEnvironments returns the *sysEnvRec* pointer in A0 and a result code in D0.

The *sysEnvRec*, at the moment, has the following fields (in Forth notation):

0	CONSTANT	environsVersion
2	CONSTANT	machineType
4	CONSTANT	systemVersion
6	CONSTANT	processor
8	CONSTANT	hasFPU
9	CONSTANT	hasColorQD

10	CONSTANT	keyBoardType
12	CONSTANT	atDrvVersNum
14	CONSTANT	sysVRefNum

The fields are described in IM V, but some newer configurations are not mentioned in the book. As pointed out earlier, *environsVersion* can be 1 or 2 (according to my experimentation), but the information returned is the same in both cases. *machineType* can be 0 to 7, because in addition to the MacII, we have the Mac Ix, the MacII 'small version', and the SE/030 now. *processor* can be 0 to 4, the last value corresponding to the 68030.

[Incidentally, I know about two otherwise very good programs which check the processor type, but don't think about Motorola's upward compatibility. TMON at first wouldn't run on my Ix, because it doesn't know the processor type returned by SysEnvironments. There are processor-specific resources MonC and MonI in TMON, and the 68030 requires MonC ID=3 and MonI ID=3 to be present. If one simply duplicates the MonC ID=2 and MonI ID=2 resources and changes their IDs to 3, TMON runs. Languages Systems Fortran, too, has its problems. If you compile a program with the 68020 code option on, it won't run on a Ix, because that machine has no 68020! Obviously, LS Fortran should check for 68020 or greater, and not for 68030 only. They know about that bug and will revise it.]

The SysEnvironments trap glue code, and some words returning individual values from the *sysEnv* record, are printed in the example. The last part contains some code that you may execute from the Mach2 environment and which will print the system configuration information directly to the console window. So now you know what machine you're using (isn't that great).

The CDEV

To put the SysEnvironments information to some practical use, we'll write a control panel utility (CDEV) that displays this information in the control panel window. CDEVs are described in IM V-323, and Steven Sheets has also written an article about them (MT V3#10, p.59). I'll briefly summarize the relevant points.

A control panel utility is a file of type 'cdev' and arbitrary creator (we use JLMT, for obvious reasons. *Not* even registered with Apple yet!). The file contains BNDL, ICN# and FREF ID=-4064 resources, and a JLMT ID=0 resource, to make its icon correctly appear on the desktop. The resources actually relevant to the control panel are:

mach ID=-4064, nrct ID=-4064, DITL ID=-4064, and cdev ID=-4064. 'mach' determines which machine the CDEV can run on; the control panel will display the CDEVs icon on the left side of the panel only on machines corresponding to the number

contained in 'mach' (see IM for details). The 'nrct' resource contains a number of rectangles that are drawn in the right part of the control panel when the CDEV is opened. These rectangles are filled with white and framed with a 2-pixel thick black border. The rest of the control panel is filled with light gray. After the rectangles, the dialog items in the DITL resource are drawn. The contents of the DITL are appended to the control panel's own DITL. Later when a dialog item on the left hand side of the panel is referenced, its item number will have to be incremented by the number of items in the control panel's own DITL.

'cdev' contains the actual code for the control panel handler. It is a code resource with its entry point at the very beginning (like XCMDs, MDEFs, etc.), and it is called like a Pascal function with the following parameters:

```
FUNCTION cdev(  
message,item,numItems,CPanelID: INTEGER;  
VAR theEvent:EventRecord; cdevValue: LONGINT;  
CPDialog: DialogPtr):LONGINT;
```

Our Forth code has to go through the usual gymnastics to be called correctly. We jump from the start of the resource to a piece of glue code which saves the old registers, sets up local Forth and loop return stacks, and moves the parameters from the A7 stack to the Forth stack. The main cdev routine is then called with the parameters on the Forth stack (see the listing). *message* will tell the routine what to do on this call. *item* is the item number of the dialog item that was hit; this parameter is only relevant for the hitDev message. *numItems* is the number of items that precede the CDEVs DITL in the item list. To address a dialog item from its own DITL, the CDEV has to add this number to the item number. *CPanelID* is the resource ID of the control panel DA. *theEvent* is a pointer to an event record, valid for hitDev, nulDev, activDev, deActivDev and keyEvtDev messages. *cdevValue* contains the return value of the last call of the cdev routine. A CDEV might, for instance, allocate its private storage and pass a handle to it as a function result; the next time it is called, it will find this handle back in the *cdevValue* parameter. *CPDialog* is the dialog pointer to the control panel dialog.

CDEV messages

A CDEV can receive several different messages in the *message* parameter. For our purpose, the *initDev* message is the only relevant one. This message is sent once when the CDEV is opened. All we need to do is to find the system information at this time and fill in the dialog items so they can be displayed. Once this is done, updating of the dialog will be handled by the control panel. Other CDEVs might need to handle more messages. The allowed values are:

```
initDev=0, hitDev=1, closeDev=2, nulDev=3, updateDev=4, activDev=5,  
deActivDev=6, keyEvtDev=7, macDev=8, undoDev=9,  
cutDev=10, copyDev=11, pasteDev=12, clearDev=13.
```

hitDev corresponds to a mouse down event in control panel dialog item. *closeDev* is sent when the control panel is closed, *updateDev* to *keyEvtDev* are sent to handle events, and *undoDev* to *clearDev* are sent when the corresponding Edit menu item is selected.

nulDev works like the *accRun* message in a desk accessory; it is sent periodically and allows the CDEV to perform actions like cursor blinking, time updating, etc.

macDev is sent optionally on initialization, depending on the value of the 'mach' resource; when this message is sent, the control panel should check whether it can really run on this particular machine configuration.

Our CDEV routine only handles the *initDev* message. It will beep once, and then put string constants corresponding to machine type, system version etc. in the various dialog items. It has been checked on a Mac 512KE, Mac Plus, Mac SE, SE/030, Mac II and IIX. It's up to you to add the checking code for the 64K ROM Mac and Lisa.

Language Systems Fortran and Prototyper

Even though this is 'officially' a Forth column, I'd like to tell you some more of my experiences with Language Systems Fortran. That system, together with Prototyper, gives you an amazingly powerful environment to outfit Fortran programs with a nice Mac-ish interface. And you can even have lengthy calculations run in the background, if you modify your Fortran code slightly.

I won't describe the program that I've been working on in detail. Suffice it to say that it's quite long, used for some kind of curve fitting, and uses some 2-5 seconds per iteration on a Microvax II. It compiles nicely under MPW/LS Fortran, and the resulting application runs from a dumb terminal window and even gives the same results as the Vax. However. Once started, Fortran takes control and releases it only at the very end of the program. 50 iterations, 2 seconds per iteration = 100 seconds of twiddling thumbs.

There is an easy way to give *some* Multifinder compatibility to LS Fortran applications. Listing 2 shows a small routine that calls *WaitNextEvent* with an event mask corresponding to update events and app4 events (context switching). Putting calls to this routine in strategic places of your Fortran program, so that it is called several times per second, will make the program 'background-able'. A mouse click on another application's window will switch the Fortran program to the background, where it will continue to execute. However, your word processing, or spreadsheet activity, is slowed down only slightly. My *pause* routine will also set the cursor to a watch when the button is pressed inside the Fortran window. That way, the program can indicate that it is still calculating, and reset the cursor to an arrow when one iteration is done; other routines could at that point grab the mouse down event and handle it, to change parameters of the calculation etc.

The second step of making a Fortran program 'Mac-ish' is to add a real menu/dialog interface for entering and changing the parameters to the program. Typically, one would enter file names, initial values, matrix sizes, etc. In my particular case, I used Prototyper to generate the interface, which gave me Pascal code. The Fortran program was split up in convenient subroutines, which were called as Externals from the Pascal main program.

The strategy was the following: Input parameters, file

names, etc. were set via menus and dialogs. When all parameters had been set, a menu item could be selected to start the calculation. This menu selection would simply set a flag in the Pascal program, *doCalc*. The main event loop's null event handler would check this flag and if it was set, run one iteration of the Fortran code per event loop. Of course, since each iteration took several seconds, this would slow down the user interaction way too much.

Therefore, whenever a non-null event was detected in the main event loop, another flag was set which disabled null event background processing. This flag was kept on for 5 seconds after each non-null event.

In summary, this strategy will transform your Fortran program into an application that can be juggled into the background at any time with 0.1 sec delay and calculate in the background. When you click on the application in the foreground, the cursor will change to a watch, the current iteration will finish, and then you can change parameters, stop the calculation, print intermediate results, etc. through the interface provided by Prototyper. When no user action is detected for 5 seconds, the program will start calculating again.

French Hacker Story - the End (???)

You might have followed the story of Faraglace: The French hacker who was supposed to pay very heavily for the mistake of having removed the copy protection from some programs - notably 4th Dimension. He did this out of his own private curiosity, the products of this activity got into the hands of some pirates, and the whole issue was taken to court. It seems, after the result of a recent lawsuit last month, that the story is finally coming to a more reasonable conclusion: The total amount of fines and damages has gone down from a sum of the order of \$100,000 to an amount of more like \$2,000 (!). This does not settle the issue for the pirates who sold the de-protected stuff; that's separate. But the fact that a court has recognized that cracking and stealing software are not quite the same thing relieves me. ACI is still going to appeal this verdict. More to come, hopefully good news.

Till next month, happy threading

Listing 1: SysConfig cdev

```
\ System configuration cdev
\ Example for MacTutor written in Mach2 Forth
\ J.Langowski March 1989
```

only forth also mac also assembler

```
\ Define SysEnviroms trap; not present in Mach2.14
\ alternatively, use the trap compiler accessible on the
\ 68010 Mach2 libraries and on the V5#1 source code disk
```

```
.TRAP _SysEnviroms $A000
```

```
0 CONSTANT environsVersion
2 CONSTANT machineType
4 CONSTANT systemVersion
6 CONSTANT processor
8 CONSTANT hasFPU
9 CONSTANT hasColorQD
10 CONSTANT keyBoardType
```

```
12 CONSTANT atDrvrsVersNum
14 CONSTANT sysVRefNum
```

```
\ compiler support words for kernel-independent
\ definitions, defproc resources, etc.
\ :xdef compiles a JMP at the beginning of the
\ block, which is resolved at the end of the definition
\ by ;xdef.
```

```
: :xdef ( - branch marker )
  create -4 allot
  $4EFA w, ( JMP )
  0 w, ( entry point to be filled later )
  0 , ( length of routine to be filled later )
  here 6 - 76543 ( marker for stack checking )
;

; :xdef ( branch marker entry | - )
  marker 76543 <> abort" xdef mismatch"
  entry branch - branch w!
  here branch - 2+ branch 2+ !
;

: xlen 4 + @ ; ( get length word of external definition )

\ **** cdev proc glue macros for kernel-independent code
```

```
CODE cdev.prelude
  LINK A6, #-512 ( 512 bytes of local Forth stack )
  MOVEM.L A0-A5/D0-D7, -(A7) ( save registers )
  MOVE.L A6, A3 ( setup local loop return stack )
  SUBA.L #256, A3 ( in the low 256 local stack bytes )
  MOVE.L 8(A6), D0 ( CPDialog )
  MOVE.L 12(A6), D1 ( cdevValue )
  MOVE.L 16(A6), D2 ( theEvent )
  CLR.L D3
  MOVE.W 20(A6), D3 ( CPanelID )
  EXT.L D3 ( in case this is negative )
  CLR.L D4
  MOVE.W 22(A6), D4 ( numItems )
  CLR.L D5
  MOVE.W 24(A6), D5 ( Item )
  CLR.L D6
  MOVE.W 26(A6), D6 ( message )
  MOVE.L D6, -(A6)
  MOVE.L D5, -(A6)
  MOVE.L D4, -(A6)
  MOVE.L D3, -(A6)
  MOVE.L D2, -(A6)
  MOVE.L D1, -(A6)
  MOVE.L D0, -(A6)
  RTS \ just to indicate the MACHro stops here
END-CODE MACH
```

```
CODE cdev.epilogue ( resCode - )
  MOVE.L (A6)+, D0
  MOVE.L D0, 28(A6) ( store function result )
  MOVEM.L (A7)+, A0-A5/D0-D7 ( restore registers )
  UNLK A6
  MOVE.L (A7)+, A0 ( return address )
  ADD.W #20, A7 ( pop off 20 bytes of parameters )
  JMP (A0)
  RTS
END-CODE MACH
```

```
\ the actual cdev code starts here.
\ REMEMBER: don't use CALL for the toolbox routines;
\ use (CALL) instead, which is not dependent on D4
\ pointing to a correct stack.
```

```
:xdef myCdev

\ just to put some text into the resource
\ for easier identification
```



```

: start " Mach2 Forth cdev example, JL/MacTutor 1989" ;

CODE SysEnvirons
  ( theWorld versrequested - theWorld resCode )
  MOVEA.L 4(A6),A0
  MOVE.L (A6)+,D0
  ADDQ.L #4,A6
  _SysEnvirons
  MOVE.L A0,-(A6)
  EXT.L D0
  MOVE.L D0,-(A6)
  RTS
END-CODE

\ words which extract single items from the SysEnvRec

: ?mach ( | [ 12 lallot ] sysEnvRec - machine# )
  ^ sysEnvRec 1 SysEnvirons drop 2+ w@ ;

: ?sys ( | [ 12 lallot ] sysEnvRec - system# revision# )
  ^ sysEnvRec 1 SysEnvirons drop
  dup 5 + c@ swap 4+ c@ ;

: ?proc ( | [ 12 lallot ] sysEnvRec - machine# )
  ^ sysEnvRec 1 SysEnvirons drop 6 + w@ ;

: ?fpu ( | [ 12 lallot ] sysEnvRec - machine# )
  ^ sysEnvRec 1 SysEnvirons drop 8 + c@ ;

: ?colorQD ( | [ 12 lallot ] sysEnvRec - machine# )
  ^ sysEnvRec 1 SysEnvirons drop 9 + c@ ;

: ?keyType ( | [ 12 lallot ] sysEnvRec - machine# )
  ^ sysEnvRec 1 SysEnvirons drop 10 + w@ ;

: ?atkVers ( | [ 12 lallot ] sysEnvRec - machine# )
  ^ sysEnvRec 1 SysEnvirons drop 12 + w@ ;

: ?sysVRef ( | [ 12 lallot ] sysEnvRec - machine# )
  ^ sysEnvRec 1 SysEnvirons drop 14 + w@ 1_ext ;

\ factored out the GetDItem/SetIText stuff

: set.item ( string dlgPtr #item ) ( | type hItem box - )
  ^ type ^ hItem ^ box (call) GetDItem
  hItem swap (call) SetIText ;

\ display system characteristics
\ in the cdev dialog box (DITL -4064 resource dependent)
\ the strings are hard-coded, but could as well be contained
\ in a STR# resource

: display.it ( numItems dlgPtr | [ 16 lallot ] str1 - )
  ?mach CASE
    0 OF " unknown" ENDOF
    1 OF " Mac 512KE" ENDOF
    2 OF " Mac Plus" ENDOF
    3 OF " Mac SE" ENDOF
    4 OF " Mac II" ENDOF
    5 OF " Mac IIX" ENDOF
  \ wasn't sure whether machine=6 is the new baby MacII,
  \ so left out that case
    7 OF " Mac SE/030" ENDOF
    " NEW MACHINE"
  ENDCASE  dlgPtr 3 numItems + set.item

\ get system version #
\ and convert to string, format X.XX
\ if you don't know Forth, this might be hard to read :- )
?sys
  ^ str1 swap (call) numtostring
  dup c@ 1+ + swap (call) numtostring
  dup c@ 1 = IF dup 1+ c@ over 2+ c!

```

```

    dup 1+ ascii 0 swap c! THEN
  ascii . swap c!
  ^ str1 dup c@ 3 + swap c!
  ^ str1 dlgPtr 5 numItems + set.item

?proc CASE
  0 OF " unknown" ENDOF
  1 OF " 68000" ENDOF
  2 OF " 68010" ENDOF
  3 OF " 68020" ENDOF
  4 OF " 68030" ENDOF
  " NEW"
ENDCASE  dlgPtr 7 numItems + set.item

?fpu IF " yes" ELSE " none" THEN
  dlgPtr 9 numItems + set.item

?colorQD IF " yes" ELSE " no" THEN
  dlgPtr 11 numItems + set.item

?keyType CASE
  0 OF " unknown type" ENDOF
  1 OF " 'old' Macintosh keyboard" ENDOF
  2 OF " 'old' Macintosh keyboard with keypad"
  ENDOF
  3 OF " Macintosh Plus keyboard" ENDOF
  4 OF " Apple Desktop Bus extended keyboard"
  ENDOF
  5 OF " Apple Desktop Bus standard keyboard"
  ENDOF
  " something NEW"
ENDCASE  dlgPtr 13 numItems + set.item

  ^ str1 ?atkVers (call) numtostring
  dlgPtr 15 numItems + set.item
;

: testCdev ( message item numItems CPanelID
  theEvent cdevValue CPDialog - result )
  \ we only need to respond to the initDev message
  \ by putting the system configuration info
  \ into the cdev's dialog items
  message CASE
    0 OF ( initDev ) 1 (call) sysbeep
      numItems CPDialog display.it ENDOF
    ( insert handlers for other messages here )
  ENDCASE
  cdevValue \ everything OK: return old cdevValue
;

: cdev.glue
  cdev.prelude
  testCdev
  cdev.epilogue
;

' cdev.glue ;xdef

\ end of cdev code

\ making the cdev resource, the usual way

: $create-res call CreateResFile call ResError L_ext ;

: $open-res ( addr | refNum - result )
  addr call openresfile -> refNum
  call ResError L_ext
  dup not IF drop refNum THEN
;

: $close-res call CloseResFile call ResError L_ext ;

: make-cdev ( | refNum - )

```

```

" cdev.res" dup $create-res
abort" You have to delete the old 'cdev.res' file first."
$open-res dup -> refNum call UseResFile
['] myCdev dup xlen
  call PtrToHand drop ( result code )
  ASCII cdev -4064 " cdev JL" call AddResource
  refNum $close-res drop ( result code )
;

\ End of cdev creation code.
\ Following are some words that can be executed
\ from within the Mach2 system, and output
\ system configuration information directly to the console.

: myMachine cr ." This is a"
?mach CASE
  0 OF ." n unknown machine," ENDOF
  1 OF ." Mac 512KE," ENDOF
  2 OF ." Mac Plus," ENDOF
  3 OF ." Mac SE," ENDOF
  4 OF ." Mac II," ENDOF
  5 OF ." Mac IIX," ENDOF
  7 OF ." Mac SE/030," ENDOF
  ." NEW MACHINE,"
ENDCASE
;

: mySystem ?sys
  ." running system v. "
  <# # #> type ascii . emit
  <# # #> type ascii . emit
;

: myProcessor cr ." It uses a"
?proc CASE
  0 OF ." n unknown" ENDOF
  1 OF ." 68000" ENDOF
  2 OF ." 68010" ENDOF
  3 OF ." 68020" ENDOF
  4 OF ." 68030" ENDOF
  ." NEW"
ENDCASE
  ." processor"
;

: myFPU ?fpu IF
  ascii , emit cr
  ." and has an arithmetic coprocessor installed."
ELSE
  ascii . emit
THEN
;

: myCQD cr ." Color QuickDraw is "
?colorQD 0= IF ." not " THEN
  ." available."
;

```

```

: myKeyBoard cr ." The Keyboard is "
?keyType CASE
  0 OF ." of an unknown type." ENDOF
  1 OF ." the 'old' Macintosh type." ENDOF
  2 OF ." the 'old' Macintosh type with keypad." ENDOF
  3 OF ." the Mac Plus type." ENDOF
  4 OF ." the ADB extended type." ENDOF
  5 OF ." the standard ADB type." ENDOF
  ." a NEW type."
ENDCASE
;

: myAtkDrvr cr ." Appletalk v. " ?atkVers .
  ." is installed."
;

: machTest
  myMachine mySystem
  myProcessor myFPU
  myCQD
  myKeyBoard
  myAtkDrvr
  cr
;

```

Listing 2: Language Systems Fortran background enabler

```

      subroutine pause(sleep)
!!m Inlines.f
      integer*4 sleep
      c
      c call Waitnextevent to generate a pause.
      c sleep parameter in ticks.
      c
      c will discard all Multifinder events (update, Suspend/Resume)
      c and set cursor to watch if button is pressed
      c
      include '::fincludes:Memtypes.f'
      include '::fincludes:Quickdraw.f'
      include '::fincludes:OSIntf.f'
      include '::fincludes:Toolintf.f'
      record /eventrecord/ myEvtRec
      record /CursHandle/ myCurs
      logical*1 p

      p = waitnextevent (%val(updateEvt+app4Evt),
        %ref(myEvtRec), %val(sleep), %val(0))

      if (button) then
        myCurs.CrsrH = getCursor(%val(watchcursor))
        call setCursor(%val(myCurs.CrsrH*.CrsrP))
      end if

      return
      end

```



Forth Forum

INITs and Patches



Jörg Langowski
MacTutor Editorial Staff
Volume 5, Number 6

"INITs and trap patches"

This month's program started off as an attempt to write a 'logging' utility, that is, a routine that will record when applications are launched and closed. Simple, I thought. Just patch the `_Launch` and `_ExittoShell` traps and install some code that writes a line to a file giving the time, date and name of the application each time these traps are called. However, I soon realized that it's not as simple as that. Under Finder, yes, that strategy works; but imagine my surprise when I found that `_Launch` is never called when an application is started under Multifinder! This 'feature' seems not to be documented in the technical notes that I have, nor in the Multifinder documentation.

If you record trap calls with TMON, you'll find the following sequence for launching an application under Finder:

```
_Launch  
_LoadSeg  
_LoadSeg  
... (until all necessary segments are loaded)
```

and for exiting, `_ExittoShell` is called. Multifinder seems to use the same sequence of calls, except for the first call to `_Launch`, which is left out.

Therefore, if we want to build a routine that logs application launches, we have to find another indicator. One trap call is always made when an application is started: `_GetResource` is called for CODE ID=0. If we patch `_GetResource` such that this particular call is intercepted, we have a 'hook' that allows us to install any routine to be called at application startup.

Patching traps under Multifinder

Traps cannot be easily patched in a permanent way under Multifinder. Since different application might need different trap patches, Multifinder intercepts (I believe) `_SetTrapAddress` and switches the trap patches when control is transferred from one program to another. So, when you write a short application that patches a trap, that patch will only be valid in the context of that application - and disappear when the application quits again. Not very useful for a routine that logs application launches, which will have to work independently of any particular context. So we'll have to install the patch before Multifinder is started, and this is best done with an INIT.

This month's example will show you how to write an INIT under Mach2 that installs permanent patches to `_GetResource` and `_ExittoShell`. For the moment, all these patches do is beep once when the application starts and when it quits. These patches would then have to be extended to write some text to a file that gives the name and time of the application.

I'll also give an example of a small application that can be used to run an INIT from a file in the same folder as the

application. It uses the same mechanism as the system itself to call the INIT. This program is based on an idea of Eric Carrasco from the Calvacom bulletin board in Paris, who wrote a similar program in Pascal.

The INIT

The mechanism by which an INIT resource is called has been outlined already by Bob Denny in V1#9. In case you're missing that issue, I'll quickly repeat it here: First, the INIT resource is loaded into the system heap. Then, `_DetachResource` is called which tells the resource manager to forget this resource. Finally, the system jumps to the first location of the INIT resource via a JSR. When the INIT later returns via RTS, the first two words in its code are overwritten by NOP instructions. The reason for this procedure, which might seem a little strange, is that an INIT code might consist of a JMP to some installation code at the end, and have the actual memory-resident part at its beginning. That way, when the JMP at the beginning is overwritten by NOPs, any code that jumps to the start of the INIT will now execute its main part. The initialization part at the end can be given back to the memory manager, once the INIT is installed.

Listing 1 shows the INIT code in Mach2. The kernel-independent code is produced in the usual way, using the external definition words which make sure the JMP at the first location if the INIT points to the right piece of code, and which set up the local Forth stack.

The INIT (word `INITRun`) first creates a handle to its own memory block by calling `_RecoverHandle`, then locks this handle. A temporary area for the Quickdraw globals is then created in the local variable space. Remember that local variables in Mach2 are allocated from top to bottom in memory, so that the sequence

```
newA5 myGlobals [ 202 1allot ]
```

with A5 later pointing to the address of newA5 will have the QD globals allocated correctly below A5. If your INIT needs to use any of them, simply reference them starting at myGlobals.

Our INIT code sets up A5 and the low memory global `CurrentA5`, then calls the usual manager initialization traps necessary for setting up a dialog box. After these calls, the environment is ready for calling the main Forth code of the INIT, `myINIT` in our case. From `myINIT`, all toolbox traps may be used since A5 points to valid QD global space and the managers have been initialized (oh... not the menu manager, of course, but you probably wouldn't want to use menus from an INIT. I doubt whether it would be possible, anyway. Try it out).

For an INIT that is supposed to stay in the system heap 'as

is', you don't need to take any special action in the code to free its memory space. However, when - as in our case - the INIT copies part of itself to a non-relocatable block in the system heap and is not needed anymore, we want to purge the memory space after action. This is done by calling `_DisposHandle` after `myINIT` has returned.

`gINIT` is the glue code that calls the standard 'INIT interface' `INITrun`. Here, the registers are saved, stacks are set up, and `CurrentA5` is restored after `INITrun` has returned.

The main INIT code, `myINIT`, first moves the patch code into a non-relocatable block in the system heap. It then patches `_GetResource`, using `_SetTrapAddress`, and moves a vector to the trap's old address into the appropriate location inside the patch. The user is notified of the patch by an alert. The `_GetResource` patch itself (`GetResPatch`) checks whether this trap has been called with resource type CODE, ID=0. In that case, it beeps once. Instead of the beep, one could insert some code that writes a line to a file which has been opened in the installation part of the INIT. Something left as an exercise...

There is also a patch to `_ExittoShell`, `ExitPatch`, which is supposed to beep when an application quits. Now look at the code where the INIT installs that patch. We do a `_GetTrapAddress`, right, but then the ugly part starts. If `_ExittoShell` is patched by `_SetTrapAddress` at system INIT time, the patch gets installed correctly, and you hear the beep when the Finder quits and calls the Multifinder, but then, silence. The patch has disappeared. Multifinder patches `_ExittoShell` as well and probably doesn't conserve previous patches. I don't know whether this is a bug or a feature (such as not calling `_Launch` when starting an application); but it makes life a great deal more difficult. Apple, can you hear me?

Here's the workaround that I thought up, which works under System 6.0.2, and is not guaranteed to work under a new release (no, don't leave the room yet, PLEASE!). When Multifinder patches `_ExittoShell`, it installs a JMP instruction to its patch code in RAM. Our INIT code tests whether the pointer returned by `_GetTrapAddress` points to a JMP <absolute> instruction. In that case, it patches the next two words with a pointer to our patch code and saves the old JMP address in the patch code. If the first instruction is not a JMP <absolute>, we assume we're running under Finder and use `_SetTrapAddress` to patch the trap. Ugly, isn't it. The worst part is that if you run under Multifinder your patch will be installed and then removed again; then you have to run the INIT again under Multifinder to install the `_ExittoShell` patch permanently. But I couldn't find an easier way, please tell me if you have one.

How can you run an INIT at all when Multifinder runs already? This is where the short `runINIT` application comes in (Listing 2). This program will simulate the way the system calls an INIT resource. It will locate the file 'theINIT' which must be in the same folder, get the INIT ID=12 resource from that file and execute it the same way the system boot code would do it. So for installing the `_ExittoShell` patch under Multifinder, you simply double-click `runINIT` in the system folder, or make it one of your startup applications.

INIT Incompatibilities

Our INIT is also a practical case where we can document that frequent disease of Mac system folders, INIT incompatibility. This INIT calls all the toolbox managers necessary to set up an alert, and then actually displays one. That means, `_WaitNextEvent` will be called from the alert. Other INITs might patch `_WaitNextEvent`, `_GetNextEvent` or `_SystemTask` to take some action when the system is fully booted up. That way, for instance, the 'Remember' INIT seems to call its associated desk accessory when some important dates are coming up. The problem is, when Remember is executed before an INIT that displays a dialog, the patch will open the desk accessory, which produces disaster in form of a bomb box.

I guess this incompatibility will always occur when you have one INIT which patches `_SystemTask` or similar routines to execute some other application or DA later on. If another INIT happens to call the patched trap and the system is not ready yet for whatever the first INIT wanted to be executed, a crash will probably follow. Make sure that this INIT is executed before any INITs that patch `_WaitNextEvent`, `_GetNextEvent` or `_SystemTask`.

Mach2 news

Finally I want to advertise the GENIE Mach2 and FIG round tables again. There is simply more interesting stuff in the software libraries than I could ever discuss here; the Mach2 system has been extended by hundreds of useful words already (by the way, v2.15 is on its way), and you find more in the FIG Forth libraries. Especially with the ANSI Forth standard coming up, a lot of the discussion on that new standard - and its implications on Mach2 - takes place on GENIE. For those of you who can't access GENIE, but have access to Bitnet directly or via a gateway, I'll try to get a compilation of the most important new words to you if you send me electronic mail:

LANGOWSKI@FREMBL51.BITNET

Happy threading.

Listing 1: Trap patch INIT

```
\ INIT example which patches _GetResource
\ with a call to _Sysbeep if type=CODE id=0
\ Also patches _ExittoShell, using an absolutely
\ ANFUL hack, but a _SetTrapAddress patch
\ seems to be removed under Multifinder
\ J. Langowski / MacTutor April 1989
only forth also mac also assembler
( *** compiler support words for external definitions *** )
: :xdef
  create -4 allot
  $4EFA w, ( JMP )
  0 w, ( entry point to be filled later )
  0 , ( length of routine to be filled later )
  here 6 - 76543
;

: ;xdef ( branch marker entry | - )
  marker 76543 <> abort" xdef mismatch"
  entry branch - branch w!
  here branch - 2+ branch 2+ !
;

: :xlen 4 + @ ; ( get length word of external definition )
```

```
\ **** ext procedure glue macros
```

```
CODE ext.prelude
```

```
LINK A6, #-700 ( 700 bytes of local Forth stack )
MOVE.M L A0-A5/D0-D7, -(A7) ( save registers )
MOVE.L A6, A3 ( setup local loop return stack )
SUBA.L #500, A3 ( in the low 200 local stack bytes )
RTS \ just to indicate the MACHro stops here
END-CODE MACH
```

```
CODE ext.epilogue
```

```
MOVE.M L (A7)+, A0-A5/D0-D7 ( restore registers )
UNLK A6
RTS
END-CODE MACH
```

```
.trap _newPtr, SYS$A51E
```

```
-4 CONSTANT thePort
$904 CONSTANT CurrentA5
$A9A0 CONSTANT tGetRes \ GetResource
$A9F4 CONSTANT tExit \ ExitToShell
```

```
\ | _____ |
\ | INIT resource code starts here |
\ | _____ |
```

```
:xdef beeperINIT
```

```
header PatchStart
```

```
header oldGetRes
```

```
DC.L 0
```

```
header oldExit
```

```
DC.L 0
```

```
: GetResPatch
ext.prelude
CLR.L D0
MOVE.W 8(A6), D0
MOVE.L 10(A6), D1
MOVE.L D0, -(A6)
MOVE.L D1, -(A6)
```

```
\ main FORTH code starts here
\ this can be used to log any launch
\ (i.e. GetResource CODE 0 ) to a log file
\ which has to be created/opened by the INIT code
```

```
ascii CODE = swap 0= and IF
\ (call) debugger
1 (call) sysbeep
```

```
THEN
\ end of main code
```

```
ext.epilogue
LEA oldGetRes, A0
MOVE.L (A0), A0
JMP (A0)
```

```
: ExitPatch
ext.prelude
```

```
\ main FORTH code starts here
\ this can eventually be used to write a line to
\ the same log file as before
\ (call) debugger
1 (call) sysbeep
\ end of main code
```

```
ext.epilogue
LEA oldExit, A0
MOVE.L (A0), A0
JMP (A0)
```

```
;
```

```
header PatchEnd
```

```
: movePatch ( | length - patch )
['] patchEnd ['] PatchStart -> length
length
MOVE.L (A6)+, D0
_newPtr, sys
MOVE.L A0, -(A6)
dup IF ( we have space in system heap )
['] PatchStart over length swap
(call) blockMove drop
THEN
```

```
: myINIT ( | patch pExit - )
movePatch -> patch
patch IF
\ patch _GetResource
tGetRes (call) GetTrapAddress
patch ! \ old GetResource
['] GetResPatch ['] PatchStart -
patch + tGetRes (call) SetTrapAddress
" GetResource patch has been installed."
0 0 0 (call) ParamText
1000 0 (call) NoteAlert drop

\ patch _ExitToShell, using hack
tExit (call) GetTrapAddress -> pExit
pExit w@ $4EF9 =
IF \ is it a JMP <absolute>
\ ? we're probably in Multifinder...
pExit 2+ 0
patch 4+ ! \ old ExitToShell
['] ExitPatch ['] PatchStart -
patch + pExit 2+ !
\ patch directly into Juggler's innards. B00!
" ExitToShell patch in Multifinder."
0 0 0 (call) ParamText
1000 0 (call) NoteAlert drop
ELSE
pExit patch 4+ !
['] ExitPatch ['] PatchStart -
patch + tExit (call) SetTrapAddress
" ExitToShell patch in Finder."
0 0 0 (call) ParamText
1000 0 (call) NoteAlert drop
THEN
ELSE
" Can't get memory for patches."
0 0 0 (call) ParamText
1000 0 (call) NoteAlert drop
THEN
```

```
: INITRun ( | newA5 myGlobals [ 202 1allot ] theHandle - )
\ (call) debugger
['] beeperINIT (call) recoverHandle -> theHandle
theHandle (call) Hlock drop
^ newA5
MOVE.L (A6)+, A5 \ create area for QD globals
MOVE.L A5, CurrentA5 \ A5 points to it
^ newA5 thePort + (call) InitGraf
(call) InitFonts
(call) InitWindows
(call) TEInit
0 (call) InitDialogs
(call) InitCursor
```

```
myINIT \ call main INIT routine
```

```
theHandle (call) HUnlock drop
theHandle (call) DisposHandle drop
```

```

;
: gINIT
  ext.prelude INITrun ext.epilogue
  MOVE.L A5,CurrentA5
;
' gINIT ;xdef
( *** creating the INIT file *** )
: $create-res call CreateResFile call ResError L_ext ;
: $open-res ( addr | refNum - result )
  addr call openresfile -> refNum
  call ResError L_ext
  dup not IF drop refNum THEN
;
: $close-res call CloseResFile call ResError L_ext ;
: make-init ( | refNum - )
  " theINIT" dup $create-res drop
  $open-res dup -> refNum call UseResFile
  ascii INIT 12 call GetResource
  ?dup IF call RmveResource THEN
  [' beeperINIT dup xlen
    call PtrToHand drop ( result code )
    ascii INIT 12 call GetResource
    ?dup IF call RmveResource THEN
    ascii INIT 12 " Beeper" call AddResource
    refNum $close-res drop ( result code )
;

```

Listing 2: INIT runner application

\ INIT 12 starter
 \ J. Langowski / MacTutor April 1989
 \ Thanks to Eric Carrasco (Calvacom EC10) to this idea

```

\ which he implemented in Pascal
: $open-res ( addr | refNum - result )
  addr call openresfile -> refNum
  call ResError L_ext
  dup not IF drop refNum THEN
;
: $close-res call CloseResFile call ResError L_ext ;
$4E714E71 CONSTANT nopnop
: runINIT ( | refNum hINIT - )
  " theInit" $open-res -> refNum
  refNum 0 > IF
    ascii INIT 12 call GetResource -> hINIT
    hINIT IF
      hINIT call HomeResFile
      refNum = IF
        hINIT call DetachResource
        hINIT @ execute \ run the INIT
        nopnop hINIT @ !
        \ fill first 2 words with NOPs
    ELSE
      " INIT not from my file" 0 0 0 call ParamText
      1000 0 call NoteAlert drop
    THEN
  ELSE
    " Can't find INIT 12 resource"
      0 0 0 call ParamText
      1000 0 call NoteAlert drop
  THEN
  ELSE
    " Can't open file 'theINIT'" 0 0 0 call ParamText
    1000 0 call NoteAlert drop
  THEN
  bye
;

```



Forth Forum

Appletalk Protocol Handlers



Jörg Langowski
MacTutor Editorial Staff
Volume 5, Number 7

“Appletalk protocol handlers”

Many of you may have used Appletalk in one or the other of their programs, but the way it really works is an interesting mystery to most of us. Since a recent project of mine will have to use some of the low-level features of Appletalk, I'd like to describe some of the hooks built into it that allow you to set up your own network protocols or change those provided by Apple.

Long-time readers of MacTutor will remember that we had a series of articles on Appletalk in V1#10 and #11 already; also one of my Forth columns (V4#9) showed some examples how to use the Appletalk services from Mach2. All these articles were mainly dealing with the high-level features of Appletalk, ATP and higher. The way the low-level stuff works was more or less taken for granted, and that's the way 95% of all programs would normally use Appletalk. Why and when do we have to take a closer look?

Imagine, for instance, a program that implements a bridge between two Appletalk networks. One may be the Localtalk connection that your Mac is hooked up to through the serial port, the other the Ethernet that you have plugged into your Ethernet card. Apple's Network CDEV lets you change from one network to the other, but you can't (yet) choose two networks simultaneously. Infosphere's LIAISON™, however, allows you to do just that, bridging Localtalk and Ethernet by a process that runs on your Mac in the background. Thus, there exists at least one example to show that an Appletalk bridge can be implemented as a background process on the Mac. (The other solution, of course, being a hardware box like Kinetics' FastPath or the Gatorbox). Such programs must use the Appletalk routines at a lower level; and I'll give some examples how to do that in the following.

DDP packets

Remember how internet addressing works on Appletalk: Each local network has a unique network number, each device on the network a unique node number, and each separate process on the device a unique socket number.

When a process on the Mac (e.g. a word processing program) wants to communicate with another device on the network (let's say, a printer), it will first look up its internet address using the name binding protocol described in the articles mentioned above. Once the internet address is known, it can then send out a packet to the remote device over the network or receive packets from it. The two devices can be either on the same local network, or on two different networks that are connected through bridges. Depending whether the remote device is on the same local network or not, the Appletalk driver will send your data to the network in either of two different formats. You normally don't see the difference; the datagram delivery protocol (DDP) takes

care of checking whether the destination network number is the same as your own network number or not. The two different formats are called long or short DDP packets, and their format is shown in Fig. 1.

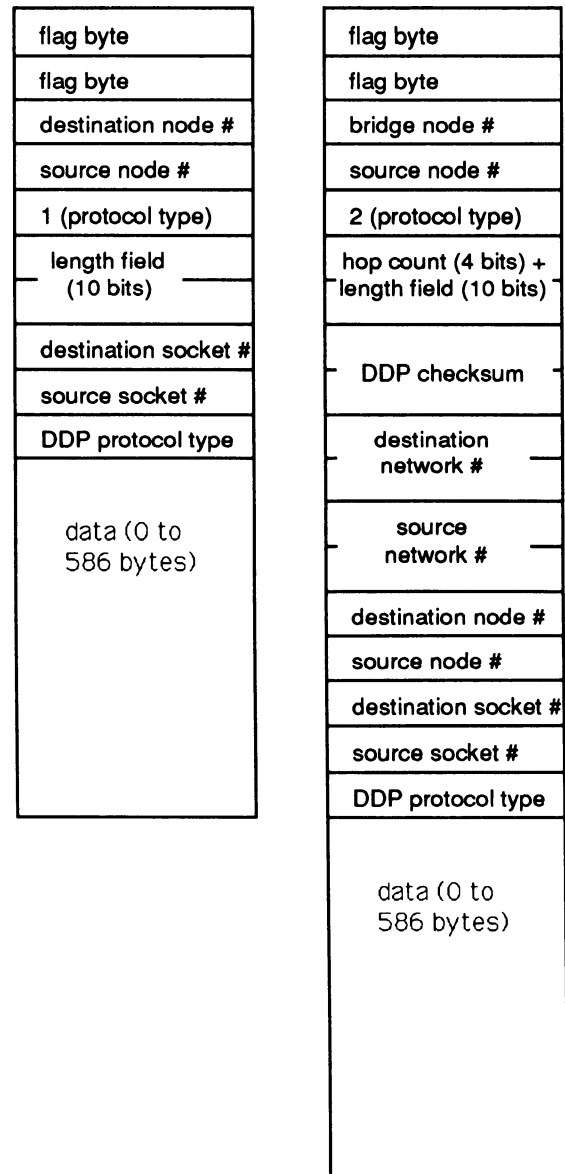


Fig.1 : short (left) and long (right) DDP packets

Typically, when the two communicating nodes are on the same local network, DDP will send short packets. When the two nodes are on different networks, DDP will send long packets; also when the two nodes are on the same network, but the sending

node doesn't know its own network number. In that case it will set the source network number of the packet to zero and send a long packet anyway.

The first two header bytes after the flag bytes (destination and source node #) determine the two nodes on the local network that communicate with each other. In a short packet, the destination node number is the number of the final node of the communication link. However, for a long packet that is routed through a bridge, the immediate destination node is that bridge; therefore the first header byte contains the bridge's node number and the final destination is given further down in the packet.

The distinction between the two packet formats is made in the third byte of the header, the LAP protocol type. LAP stands for link access protocol, the lowest level of the Appletalk protocol which delivers data from one node to the other on the same local network. The LAP protocol then determines what to do with the packet after it has been received.

Node addressing

There is a lot of traffic going on on a typical Appletalk network, and each node has to filter out only those packets that it needs to receive, that is, those where the destination node ID matches its own ID, or where the destination ID is \$FF (broadcast packets). This is a task that has to be done by dedicated hardware. We cannot expect the Macintosh CPU to look at each single packet and see whether it has the correct ID; doing that, we wouldn't have time for doing any other work.

Fortunately, the Macintosh's 8530 SCC chip (serial communications controller) can be programmed to automatically detect a flag byte - a 01111110 sequence - followed by an address byte - the destination address -, and send an interrupt only if the received address matches a preset value. That way a node will ignore all packets except those that it is actually supposed to receive.

LAP protocol handlers

Now, a packet is arriving that carries the correct node number in its header, what are we going to do with the data? Whatever we do, we should do it fast, because data is arriving at a rate of 260 KBaud. The SCC's internal buffer holds three bytes, so we have only 95 μ s to take any action required. The decision what to do with the packet depends on the value of the third header byte, the LAP protocol field. Each packet structure (DDP long/short, other structures that you might have implemented) corresponds to a different protocol number.

For each protocol number that the node understands, there will be an entry in a protocol handler table consisting of the protocol number and the handler's address. The low-level packet reception routine will search the table for the protocol number, and transfer control to the corresponding handler if it is found. Otherwise, the packet will just be ignored.

The protocol handler table is located in the Appletalk global variable area; the address of this area is kept in the low-memory global *ABusVars* (\$02D8). The structure of the Appletalk global variable area is described in part in Inside Macintosh II-328, the protocol table is not specified there. This is because the format of

the protocol table depends on the specific Appletalk implementation, the MPP driver in the System file expects a different format than that in ROM. As an example, I'll give the format for the MacII (and SE) when Appletalk has been loaded from ROM:

0	sysLAPAddr	This node ID (byte)
1	toRHA	read header area (24 bytes)
25	sysABridge	Node ID of a bridge on the local network (byte)
26	sysNetNum	This network number (word)
28	vSCCEnable	status register value to re-enable SCC interrupts (word)
..... (some unspecified bytes)		
36	LAPprotNums	LAP protocol numbers (8 bytes)
44	LAPprotProcs	LAP protocol handlers (32 bytes = 8 pointers)

Each entry in the protocol table is either a protocol number at $(36+i)$ and a corresponding address at $(44 + 4*i)$ with $i=0$ to 7, or \$FF and a zero address for free slots in the table. We see that a maximum of eight different protocols are allowed; this is because checking of the protocol type and control transfer to the handler must be completed in the allowed time frame of 95 μ s.

Thus, when the protocol number of the packet corresponds to a valid protocol handler in the table, control will be transferred to that protocol handler. This month's example program contains a protocol handler that can be installed instead of the default handler for long DDP packets (LAP type 2). A warning in advance: Installing this handler will completely screw up most of your Appletalk services, since your Mac won't understand long DDP packets correctly anymore.

The default DDP protocol, after reading the packet header (Fig. 1), gets the address of a socket listener routine from the socket table, where socket numbers and listener routine pointers are arranged similar to the protocol handlers in the protocol table. The socket table is also kept in the *ABusVars* block pointed to by A2. The default protocol handler transfers control to a socket listener if it finds one in the table. For long DDP packets, we will now replace the default handler by one that simply reads the packet data into a buffer and does nothing else. You may then - from Mach2 - look at the packet data in the buffer. To restore normal Appletalk operation, you must disable and re-enable Appletalk from the Chooser. This restores the default handlers.

The new handler code is defined in the word *myLAP2*. When a packet arrives, it will first verify whether the destination network is the local network. If so, it will read the packet data into a buffer which is located just in front of the handler code. It will then construct a short DDP packet out of the long one by stripping all the network information. This packet could then be re-sent to the network; setting the *SetSelfSend* flag to true, the same node would receive it again, this time through the LAP type 1 protocol handler. The corresponding code is commented out, since it did not work for me in this simple manner. So far, the protocol handler can only be used to look at the raw packet data. I'll keep you informed when I've found the reason why.

attach.ph and *detach.ph* are used to insert and remove

handlers in the protocol table. A handler can only be attached to a protocol type if that type is not yet present in the table; therefore to change the LAP type 2 handler we have to remove the old one, then install the new one. **change.prots** gets a block in system heap space, moves the handler code with the buffer areas into that block, and installs the new protocol handler.

This almost concludes my short introduction into low-level Appletalk stuff; a lot of the information presented here is not documented in any Apple documentation that I'm aware of and could only be found out by disassembling into ROM. Disassembly also showed me the function of two more Appletalk system globals, the procedure pointers **ATalkHk1** (\$B14) and **ATalkHk2** (\$B18). Both are active when they are non-NIL: **ATalkHk1** seems to be called on each **_Control** call to the **.MPP** driver, and **ATalkHk2** on every **writeLAP** call. **ATalkHk2**, in particular, would enable you to define alternate link access protocols, as for Ethernet, ISDN, and the like (for more detail on this, see the Alternate Appletalk Connections Reference, APDA #KNB007).

Listing 1: LAP protocol handler example

only forth also assembler

\ Appletalk LAP protocol handler example

\ 12.05.89 JL

\$904 constant currentA5

DECIMAL

12 constant ioCompletion

18 constant ioFileName

18 constant userData

24 constant ioRefNum

26 constant csCode

27 constant ioPermission

28 constant socket

28 constant protType

30 constant addrBlock

30 constant handler

9 constant mppUnitNum

mppUnitNum 1+ negate

constant mppRefNum

\ LAP defs

1 constant LAPshortDDP

2 constant LAPlongDDP

-94 constant lapProtErr

-95 constant lapExcessCollns

243 constant lapWrite

244 constant lapDetachPH

245 constant lapAttachPH

-1 constant lapOverrunErr

-2 constant lapCRCErr

-3 constant lapUnderrunErr

-4 constant lapLengthErr

\ DDP defs

5 constant ddpHdSzShort

13 constant ddpHdSzLong

1 constant ddpRTMP

2 constant ddpNBP

3 constant ddpATP

\$7F constant ddpMaxWKS

586 constant ddpMaxData

\$3ff constant ddpLengthMask

128 constant ddpWKS

-91 constant ddpSktErr

-92 constant ddpLenErr

-93 constant ddpNoBridgeErr

\ CsCode values for DDP Control calls- MPP

246 constant ddpWrite

247 constant ddpCloseSkt

248 constant ddpOpenSkt

256 constant setSelfSend

\$1FA constant pRamByte

\$1FB constant SPConfig

\$291 constant portBUse

\$2D8 constant ABusVars

\$2DC constant ABusDCE

\ ABusVars block

0 constant sysLAPAddr

1 constant ioRHA

8 constant dstNetNum

25 constant sysABridge

26 constant sysNetNum

28 constant vSCCEnable

header handler.start

header ATPblock 50 allot

header LAPblock 8 allot

header packet 586 allot

.trap _control,async \$a404

.trap _newptr,sys \$a51E

CODE myLAP2

moveq.l #ddpHdSzLong-2,D3

move.w sysNetNum(a2),D2

jsr (a4)

bne #2

cmp.w dstNetNum(a2),d2

bne #1

lea packet,a3

move.l #586,d3

jsr 2(a4)

bne #2

lea LAPblock,a0

move.b toRHA(a2),(a0) \ dest node ID

move.b toRHA+1(a2),1(a0) \ source node ID

move.b #1,2(a0) \ LAP type = 1

move.b toRHA+3(a2),3(a0) \ length field MSB

move.b toRHA+4(a2),4(a0) \ length field LSB

move.b toRHA+13(a2),5(a0) \ dest skt number

move.b toRHA+14(a2),6(a0) \ src skt number

move.b toRHA+15(a2),7(a0) \ DDP prot type

\ _debugger

\ set up parameter block for LAPwrite call

\ lea ATPblock,a0

\ move.w #mppRefNum,ioRefNum(a0)

\ move.l #0,ioCompletion(a0)

\ move.w #LAPwrite,csCode(a0)

\ lea LAPblock,a1

\ move.l a1,addrBlock(a0)

\ move.w vSCCEnable(a2),sr \ re-enable interrupts

\ _control,async

@2rts

@1moveq.l #0,d3

jmp 2(a4)

END-CODE

header handler.end

: call.mpp

mppRefNum ['] ATPBlock ioRefNum + w!

['] ATPBlock call control

;

: attach.ph (protType handler - flag)

(handler) ['] ATPBlock handler + !

```

( protType ) ['] ATPBlock protType + c!
lapAttachPH ['] ATPBlock csCode + w!
call.mpp
;
: detach.ph ( protType - flag )
( protType ) ['] ATPBlock protType + c!
lapDetachPH ['] ATPBlock csCode + w!
call.mpp
;
: set.self.send ( self_send_flag | old_flag - )
setSelfSend ['] ATPBlock csCode + w!
( flag ) ['] ATPBlock 28 + c!
call.mpp drop \ result code
['] ATPBlock 29 + c@
;
: get.sys.block
['] handler.end ['] handler.start -
MOVE.L (A6)+,D0
_newptr,sys ( get memory block in system heap )

```

```

MOVE.L A0,-(A6)
;
: change.prots ( | protPtr - )
get.sys.block -> protPtr
protPtr IF
['] handler.start protPtr
['] handler.end ['] handler.start - cmove
2 detach.ph
abort" Could not detach protocol handler"
2 ['] myLAP2 ['] handler.start -
protPtr +
attach.ph
abort" Could not attach protocol handler"
255 set.self.send drop
ELSE ." Could not get memory for protocol handler"
THEN
cr ." Buffer area is at " protPtr 50 + . cr
;

```





Basic School

Time Retrieval and Storage Techniques



Dave Kelly
MacTutor Editorial Board
Volume 5, Number 1

Time Retrieval and Storage Techniques

It's only a matter of time. Anybody see the ad for the MacTutor of the future in the November, 1988 MacTutor? The ad may sound like a joke about the 100th anniversary of MacTutor, but some of the "future" has already been invented. I did some thought on the subject of the Basic School column described in the ad, which is also the title of this column. There are two basic subjects implied by the title, time retrieval and time storage.

From a philosophical point of view you may ask, how can time be saved or retrieved? Looking at the answer from that perspective would lead us to a discussion of Einstein and his theory of relativity which is beyond the scope and intent of this column. Instead let's look at some ways which we can use time to our advantage and see how storage and retrieval fits into the scheme of things.

From a user level on the Macintosh there is one time retrieval/storage mechanism that is accessible to all "the rest of us". The example I refer to is the Alarm mechanism built into the clock chip. The date and time setting is copied ("retrieved") at system startup from the clock chip into its own low-memory location. It's stored there as the number of seconds since midnight, January 1, 1904, and is updated every second. The range of the clock chip is from January 1, 1904 through February 6, 2040 (does that mean that 51 years from now I'll need to replace my clock chip?). The low-memory location can be accessed by your program. BASIC (every brand) includes its own time retrieval command. In True Basic the TIME and TIME\$ functions are provided. TIME returns the number of seconds since midnight in tenths of seconds. The TIME\$ function returns a string that contains the time as measured by the 24-hour clock with resolution in seconds. ZBasic and MS QuickBASIC uses the TIMER function which returns the seconds since midnight (similar to the True Basic TIME function) and TIME\$ which works the same as in True Basic. The problem is that the TIMER function does not provide enough resolution. At least the True Basic TIME function provided tenths of seconds or better resolution. Since we know that other languages are able to access the time in greater resolution, we may use the same function (ROM) to retrieve the time in greater resolution. This is especially important for benchmark tests where two compiled programs are being timed and the difference is tenths of seconds or better. In benchmarks that have been done in the past, sometimes the TIME function was used and there was rounding off to either 0 or 1 second. A program running in 0 seconds is absurd, but that was the best we could do with MS Basic at the time. QuickBASIC and ZBasic both allow access to the TickCount call (Macintosh ROM). In QuickBASIC a typical benchmark might be imple-

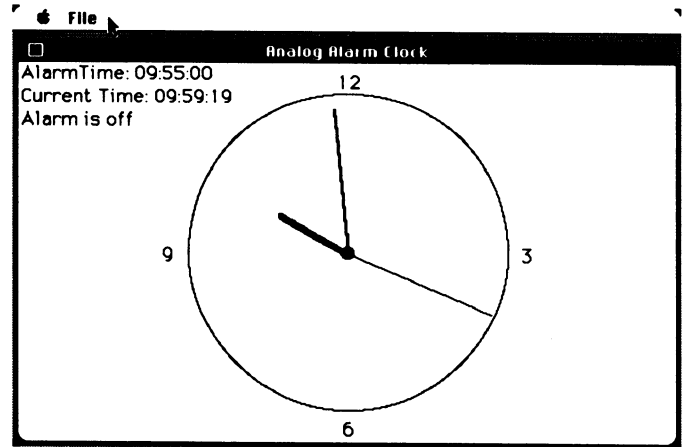


Figure 1. The Time Machine

mented such as:

First initialize the ToolBox routine:

```
TOOLBOX "I"
```

This must be used before any other ToolBox statement in the program. Now define a sub program or routine that can be called to replace the other time functions usually used in the benchmark.

```
SUB Tick (Count&) STATIC
TrapNo% = &HA975
Count& = 0% : 'Define the variable
ToolBox "L", TrapNo%, Count&
END SUB
```

Now call Tick(Count&) whenever you need the time. A tick is one-sixtieth of a second, enough for the tenth of a second accuracy provided by True BASIC's function. In ZBasic it is even easier to call TickCount. In ZBasic just use the function directly:

```
Count&=FN TICKCOUNT
```

As you can see, accessing real time is really the simple part of the job. Accessing stored time is another aspect. I'll leave the subject of how the stored time became stored for later except to say that the Parameter RAM also stores the alarm setting. You can set the alarm with the Alarm Clock DA (included with your system software).

The Analog Alarm Clock program included here demonstrates how you can retrieve the stored alarm time from the low-memory area. The Alarm time is stored at \$200. The getAlarm routine retrieves the time by peeking at memory directly. The result is then converted to date information via the secs2date

toolbox routine. The Analog Clock program reveals some problems. The date conversion library routines are used to convert seconds to dates and dates to seconds. Secs2Date works fine, but there is a problem with the Date2Secs routine. This means that the Date2Secs routine must be called a different way. Fortunately, QuickBASIC provides the ToolBox call which allows you to make ROM calls yourself. The following lines replace the Date2Secs library routine:

```
ToolBox "I"

... (Don't forget to initialize the toolbox)

TrapNo%=&HA9C7
ToolBox "R", TrapNo%, ReturnArray&(0),VARPTR(DtRec&(0))
Alt%=ReturnArray&(2)
```

It's unfortunate that we have to go through that trouble, but that's just the way it goes. I confess it took awhile before I realized that I wasn't going to be able to get it to work without the ToolBox library call.

In the program, alarm time is taken from parameter RAM and then it's updated when changed when the alarm is turned on. Be careful when poking around the parameter RAM. If the wrong parameter gets changed you may have to Zap the parameter RAM to get started. You can do this by removing your MacPlus battery for a few minutes when your power is off or by holding down command-option when opening the control panel on newer Mac systems.

Flashing the menu bar like the Alarm Clock DA does will be left for another day. Anybody know how to do that from Basic? What I'm not sure of is where the alarm status is stored. Happy New Year!

```
'Analog Alarm Clock Program
'©1989 MacTutor
'by Dave Kelly
```

```
DIM DtRec%(6),CenterRect%(3),pattern%(3)
ToolBox "I"
DIM ReturnArray&(5)
GOSUB getAlarm
false=0:true=NOT false
PI=3.14159

' Set up menus
MENU 1,0,1,"File"
MENU 1,1,1,"Turn Alarm On..."
MENU 1,2,0,"-"
MENU 1,3,1,"Quit"
CmdKey 1,3,"Q"

' Get Screen size
sheight=SYSTEM(6)
swidth=SYSTEM(5)
WINDOW 1,"Analog Alarm Clock",(4,44)-(swidth-4,sheight-4),6

' Find the center of the window, centerx%,centery%
' This should work with any size of screen
wheight%=WINDOW(3)
wwidth%=WINDOW(2)
centerx%=wwidth%/2
centery%=wheight%/2
circledia=centery%-25
```

```
GOSUB SetupClock
ON MENU GOSUB MenuEvent
ON DIALOG GOSUB DialogEvent
DIALOG ON:MENU ON
```

```
Loop:
forecolor 33:' change color to black
t$=TIME$:' Get the current time and separate into variables
hour$=LEFT$(t$,2)
Min$=MID$(t$,4,2)
Sec$=RIGHT$(t$,2)
H=VAL(hour$):M=VAL(Min$):S=VAL(Sec$)
IF H>12 THEN H=H-12
H=H*M/60

' Compute the angles for each hand of the clock
SecAngle=(360-S*6)*PI/180
MinAngle=(360-M*6)*PI/180
HAngle=(360-H*30)*PI/180

'Draw the second hand
CALL MOVETO (centerx%,centery%)
ypoint=(circledia*COS(SecAngle))*98
xpoint=(circledia*SIN(SecAngle))*98
CALL LINE (-xpoint,-ypoint)
IF SecAngle<>01dSec THEN
CALL MOVETO (centerx%,centery%)
CALL PENMODE (11)
ypoint=(circledia*COS(01dSec))*98
xpoint=(circledia*SIN(01dSec))*98
CALL LINE (-xpoint,-ypoint)
CALL PENMODE(8)
END IF

'Draw the minute hand
CALL PENSIZE(2,2)
CALL MOVETO (centerx%,centery%)
ypoint=(circledia*COS(MinAngle))*9
xpoint=(circledia*SIN(MinAngle))*9
CALL LINE (-xpoint,-ypoint)
IF MinAngle<>01dMin THEN
CALL MOVETO (centerx%,centery%)
CALL PENMODE (11)
ypoint=(circledia*COS(01dMin))*9
xpoint=(circledia*SIN(01dMin))*9
CALL LINE (-xpoint,-ypoint)
CALL PENMODE(8)
END IF

'Draw the hour hand
CALL PENSIZE(4,4)
CALL MOVETO (centerx%,centery%)
ypoint=(circledia/2*COS(HAngle))
xpoint=(circledia/2*SIN(HAngle))
CALL LINE (-xpoint,-ypoint)
IF MinAngle<>01dMin THEN
CALL MOVETO (centerx%,centery%)
CALL PENMODE (11)
ypoint=(circledia/2*COS(01dH))
xpoint=(circledia/2*SIN(01dH))
CALL LINE (-xpoint,-ypoint)
CALL PENMODE(8)
END IF
CALL PENSIZE(1,1)

forecolor 205:' Change color to red
LOCATE 1,1:PRINT "AlarmTime: ";AlarmTime$
forecolor 33:' Change color to black
LOCATE 2,1:PRINT "Current Time: ";t$
CALL TEXTMODE(0)
01dT=t$:01dSec=SecAngle:01dMin=MinAngle:01dH=HAngle
CALL FILLOVAL(VARPTR(CenterRect&(0)),VARPTR(pattern&(0)))
IF Alm=false THEN
forecolor 273
```

```

LOCATE 3,1:PRINT "Alarm is off":GOTO Loop
END IF
forecolor 341:'Change color to Green
LOCATE 3,1:PRINT "Alarm is on "
'Sound the alarm if time is up!
IF t$=AlarmTime$ THEN
    BEEP:BEEP:BEEP
    LOCATE 4,1:PRINT "Alarm event occurred"
END IF
GOTO Loop
DIALOG OFF

SetupClock:' Set up the face of the clock
activewindow=WINDOW(0)
WINDOW OUTPUT 1
CALL TEXTSIZE(14)
CALL MOVETO(centerx%-8,centery%-circledia-4)PRINT "12";
CALL MOVETO(centerx%-8,centery%+circledia+20)PRINT " 6";
CALL MOVETO(centerx%-circledia-20,centery%+7)PRINT "9";
CALL MOVETO(centerx%+circledia+10,centery%+7)PRINT "3";
PSET (centerx%,centery%)
forecolor 205:' Change color to red
CALL PENSIZE(2,2)
CIRCLE (centerx%,centery%),circledia
CALL PENSIZE(1,1)
SetRect CenterRect%(0),centerx%-5,centery%-5,centerx%+5,centery%+5
forecolor 33:' Change color to black
SetRect pattern%(0),&HFFFF,&HFFFF,&HFFFF,&HFFFF
CALL FILLOVAL(VARPTR(CenterRect%(0)),VARPTR(pattern%(0)))
WINDOW OUTPUT activewindow
RETURN

DialogEvent:
d=DIALOG(0)
SELECT CASE d
CASE 1
    IF DIALOG(1)=1 THEN editwindow=true
CASE 4
    GOTO Quit
CASE 5
    IF DIALOG(5)=1 THEN GOSUB SetupClock
CASE ELSE
END SELECT
END IF
RETURN

MenuEvent:
menunumber=MENU(0)
menuitem=MENU(1)
MENU
IF menunumber=1 THEN
    IF menuitem=1 THEN GOSUB SetAlarm
    IF menuitem=3 THEN Quit
END IF
RETURN

SetAlarm:' Set the alarm
Alarm=NOT Alarm
DoSet:
IF Alarm THEN
    MENU 1,1,1,"Turn Alarm Off"
    WINDOW 2,"", (50,50)-(250,150),2
    LOCATE 2,1:PRINT "Alarm Time:"
    IF AlarmTime$="" THEN AlarmTime$=TIME$
    EDIT FIELD 1,AlarmTime$,(100,17)-(165,32),1
    BUTTON 1,1,"OK",(40,45)-(170,75),1
    editwindow=false
    WHILE editwindow=false
    WEND
    AlarmTime$=EDIT$(1)
    a$="":i=1
    WHILE MID$(AlarmTime$,i,1)<>" "
        a$=a$+MID$(AlarmTime$,i,1)
        i=i+1

```

```

WEND
Ahour=VAL(a$)
IF Ahour>24 OR Ahour<0 THEN GOTO DoSet
a$=""
i=i+1
WHILE MID$(AlarmTime$,i,1)<>" "
    a$=a$+MID$(AlarmTime$,i,1)
    i=i+1
WEND
Amin=VAL(a$)
IF Amin>59 OR Amin<0 THEN GOTO DoSet
a$=""
i=i+1
WHILE i<=LEN(AlarmTime$)
    a$=a$+MID$(AlarmTime$,i,1)
    i=i+1
WEND
Asec=VAL(a$)
IF Asec>59 OR Asec<0 THEN GOTO DoSet
a$=""
IF Ahour<10 THEN a$="0"+STR$(Ahour) ELSE a$=STR$(Ahour)
a$=a$+":"
IF Amin<10 THEN a$=a$+"0"+STR$(Amin) ELSE a$=a$+STR$(Amin)
a$=a$+":"
IF Asec<10 THEN a$=a$+"0"+STR$(Asec) ELSE a$=a$+STR$(Asec)
AlarmTime$=""
FOR i=1 TO LEN(a$)
    IF MID$(a$,i,1)<>" " THEN
        AlarmTime$=AlarmTime$+MID$(a$,i,1)
    NEXT i
DtRec%(3)=Ahour
DtRec%(4)=Amin
DtRec%(5)=Asec
'Call Date2secs routine (QuickBASIC Date2secs does not work right)
TrapNo%=&HA9C7
ToolBox "R", TrapNo%, ReturnArray&(0),VARPTR(DtRec%(0))
A1%=ReturnArray&(2)
POKEL &H200, A1%
WINDOW CLOSE 2
ELSE
    TEXTNODE (0):LOCATE 2,1:PRINT "
MENU 1,1,1,"Turn Alarm On..."
END IF
RETURN

getAlarm:' Retrieve
A1%= PEEKL(&H200)
secs2Date A1%,DtRec%(0)
a$=""
IF DtRec%(3)<10 THEN a$="0"+STR$(DtRec%(3)) ELSE
a$=STR$(DtRec%(3))
a$=a$+":"
IF DtRec%(4)<10 THEN a$=a$+"0"+STR$(DtRec%(4)) ELSE
a$=a$+STR$(DtRec%(4))
a$=a$+":"
IF DtRec%(5)<10 THEN a$=a$+"0"+STR$(DtRec%(5)) ELSE
a$=a$+STR$(DtRec%(5))
AlarmTime$=""
FOR i=1 TO LEN(a$)
    IF MID$(a$,i,1)<>" " THEN
        AlarmTime$=AlarmTime$+MID$(a$,i,1)
    NEXT i
RETURN

Quit:
WINDOW CLOSE 1
END

```



Basic School

ADB Demo in ZBasic 5.0 & LS Pascal 2.0



Dave Kelly
David E. Smith
MacTutor Editorial Board
Volume 5, Number 3

Exploring the ADB

The Apple Desktop Bus (ADB) is a simple local-area network that connects low-speed input-only devices (for example: keyboard or mouse) to the operating system and Macintosh hardware. ADB is supported only on the Macintosh SE and Macintosh II (and future Macintosh computers). It should first be understood that the standard mouse and keyboard drivers automatically handle the ADB functions.

Communication with the ADB is accomplished with six routines located in the 256K ROM. These routines are used only if you need to access bus devices directly or communicate with a special device. This being the case, you will probably never use them. The typical access to ADB devices is through the Toolbox Event Manager routines (ex: GetNextEvent or WaitNextEvent). In BASIC, the DIALOG statement handles this process. We have discussed the DIALOG statement in detail in previous volumes of MacTutor (see Jan, 1987; Aug, 1987; Sep, 1987).

Macintosh Technical Note #206 is inspiration for this column. #206 explains how ADB works on the SE and II. The question was asked, "How can I turn on/off the LEDs on the Apple Extended Keyboard?". The answer: "Using the LEDs on the extended keyboard involves the ADBOp call (*Inside Macintosh Volume 5*). Once you determine that you have the extended keyboard, (with CountADBs, GetIndADB), then register 2 of the extended keyboard has the LED toggles in the low 3 bits. Therefore, you would do a talk to register 2 to have the device send you the contents of register 2, manipulate the low three bits to set the LEDs, and then pass the now modified register 2 back to the device with a listen to register 2 command."

The first task is determining that the extended keyboard is being used. The SysEnviroms Function is another good way to determine if the extended keyboard exists although CountADBs and GetIndADB will have to be used to get the actual keyboard address. The extended keyboard will have an ID of 2 and a device handler of 2. The standard apple keyboard will have an ID of 2 and a device handler ID of 1. Several examples of SysEnviroms have been given in previous articles. The use of the SysEnviroms function in ZBasic is shown in the Basic example in this column.

LS Pascal 2.0 Version

In addition to the ZBasic example, the same program was coded in LS Pascal 2.0 and extensively modified by David Smith, Editor and Publisher of MacTutor. This Pascal example shows how the same information displayed in the ZBasic program can be displayed using Text Edit in a fully operational text edit window. Multifinder is supported with suspend/resume events, text selection, cutting and pasting is supported, the scrap is updated properly for Multifinder, and the use of two new

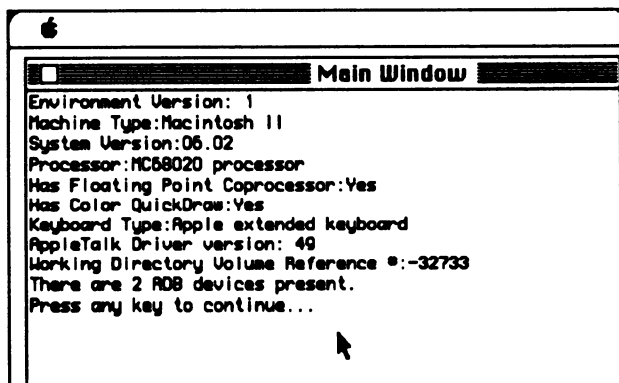


Fig. 1 Our ADB Demo in ZBasic: SysEnviroms Dump

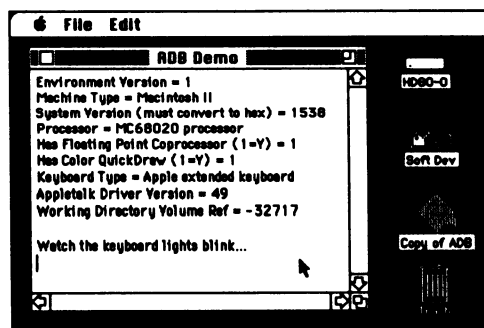


Fig. 2 Same program in LS Pascal: MultiFinder Friendly Text Edit Window!

resources, *vers* and *mstr* are used. The *vers* resources allow the version to be displayed in two places in the GetInfo box. The *mstr* resources allows MultiFinder to find the Quit command so the program can exit gracefully after a shutdown command. Both programs parse and display the SysEnviroms information about the Mac and then blink the keyboard lights if an extended keyboard is present.

ZBASIC 5.0 Version

To use SysEnviroms in ZBASIC 5.0 you will first need to modify the function using the Toolbox editor program. The SysEnviroms function is register based and so the function is modified to reflect this. When I tried the SysEnviroms function included in ZBASIC 5.0, the function did not return the correct results. Change the function to: D0.W=FN SYSENVIRON (D0.L, A0.L). This sets the result to register D0 with inputs passed to the function in the D0 and A0 registers as shown in Inside Macintosh Vol. V, pg. 6. An explanation of the SysEnviroms record is given on pg. 6-8. The source code disk for this

month includes the ADB ZBasic 5.0 library which the Toolbox editor program can move into ZBasic to create a new version of ZBasic with the correct calling sequence for Sysenvirons. The ability to extend the language to new traps or correct old ones is a very clever ability in ZBasic 5.0. It means you don't have to get a new version of the program when the toolbox calling routines or traps change.

Accessing an ADB Device

There are 16 possible device locations available on the ADB. Each device can maintain up to four variable-size registers which may be read or written from the device over the ADB network. All devices use register 0 and register 3. Register 0 is used to store input data for device interrupts. Register 3 is used for device identification and operating flags. The chapter on the ADB Manager in Inside Macintosh Vol. V explains these two registers in more detail for writing device drivers. The other two registers may or may not be used depending upon which device is being used. The Extended Keyboard uses register 2 as indicated by the comments in TN#206.

Devices which connect to the ADB contain their own micro-processors, which handle both device routines and the ADB interface. Sending commands to the device is accomplished with the ADBOp function. There are four possible commands which may be sent to the device. The ADB Command Formats are

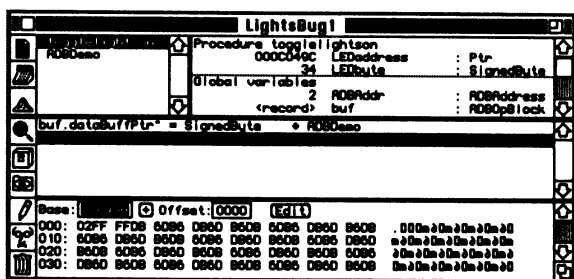


Fig. 3 Register 2 = 02FFFF after talk command

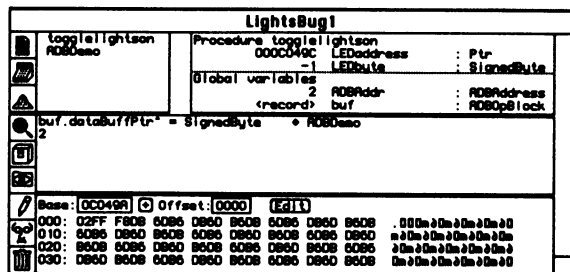


Fig. 4 Register 2 = 02FFF8 before listen command

shown on pg. 363 of Inside Macintosh. A description of the commands is given on page 364. We use the Talk command to tell the device that we want to see register 2 by sending a command of \$2E (binary '0010 1110'). The buffer pointed to by the ADBOpBlock parameter block will contain the contents of register 2 after the ADBOp function is called. As seen in the

LightsBug window the contents of the buffer after the Talk command is sent is 02FFFF. Since the technical note states that the LEDs are toggled on/off in the lower three bits, changing the contents of the buffer to 02FFF8 and sending the Listen command should turn on all three lights. The first byte of the buffer (\$02) is the number of bytes in the buffer. (See figures 3 & 4)

The Listen command is used to write the register back to the device. When the Listen command is sent after modifying the lower 3 bytes of register 2 (as stored in the buffer area), the LEDs do in fact turn on. The reverse process may be used to turn off the LEDs or sending the Flush command also clears the LED status.

The ADBOpBlock record (referred to by the ADBOp function) contains a pointer to the buffer where the registers are stored (as discussed above) and a pointer to a completion routine and data pointer. The completion routine is called by ADBOp upon completion of the function. There are three ways that a completion routine may be used. 1) You may pass a nil pointer to ADBOp when you do not wish to have a completion routine. 2) You may use the service routine address which is fetched by the GetIndADB function or GetADBInfo function. Before calling ADBOp, set the completion pointer to the service address. 3) Use your own completion routine. The data pointer is an area which the completion routine may use (optionally) to store its own data.

There is rarely a reason to call ADBOp except to tell the device something or turn on LEDs as in this example. In most cases, the ADB device communication will be handled by the system's internal polling/service request mechanism.

The two program examples included here, one in ZBasic 5.0, the other in LS Pascal 2.0, illustrate this process. The flow of the program is much easier to follow in Pascal because of the organization of the record structures. The result is the same. In order to get ZBasic to recognize the ADB Manager routines, they must be added using the Toolbox Editor. The definitions of the functions and procedures which must be added are shown below (these are included in a library file on this month's source code disk):

In the Pascal version, a complete Macintosh text edit application is given. A menu function, ADB, in the File menu, causes the doADB procedure to be called from the procedure which handles a mouse down event. When the doADB proc executes, it parses the SysEnvirons call and inserts the same information as

Mac Functions File:"ADBs Library"

(Trap-A078) DO.W-FN GETINDADB (AO.L,DO.L)
 (Trap-A077) DO.W-FN COUNTADBS
 (Trap-A079) DO.W-FN GETADBINFO (AO.L,DO.L)
 (Trap-A07A) DO.W-FN SETADBINFO (AO.L,DO.L)
 (Trap-A07C) DO.W-FN ADBOP (AO.L,DO.L)
 (Trap-A090) DO.W-FN SYSENVIRONS (DO.L,AO.L)

Mac Procedures File:"ADBs Library"

(Trap-A07B) CALL ADBREINIT

Fig. 5 ZBasic SysEnvirons and ADB Functions

the Basic program into the text edit record for the program's window. That information is then displayed as the result of an update event. This is how you would duplicate the functionality of Basic's print statements using a text edit record in a Pascal program that entirely uses the Macintosh toolbox for output into a window. Pay particular attention to the resource file at the end of the program. There in RMaker format, is the two new resources for inserting a version in the GetInfo dialog box, and for helping MultiFinder quit your application. Note also that the SIZE resource has the necessary information to tell MultiFinder that our application understands a suspend/resume event. This event, number 15, is treated much like an activate/deactivate event. By allowing for event 15, we save MultiFinder from sending us a lot of activate/deactivate events we don't need, thereby making the quasi-multi-tasking scheme work better. These two new resources are described in a recent tech note #189.

The other significant feature of the Pascal program is that it correctly allows the window to be re-sized. This requires a proper Update Event procedure and a Grow procedure. The key is that the rectangles for the text edit viewing rectangle, the scroll bars and the grow box all must be defined properly. Controls are active and updated properly, but no value was assigned the control, so it doesn't indicate scrolling even if the window is made small enough to cover some of the text. A scrolling unit is needed to match the control to the number of lines of text and the insertion point. For an example of how to do this, see the Text Edit example by David Smith from last year's MacTutor volume 3. That program had a working multiple window example of text edit with scrollers.

For more information refer to the ADB Manager chapter of Inside Macintosh and technical notes on ADB.

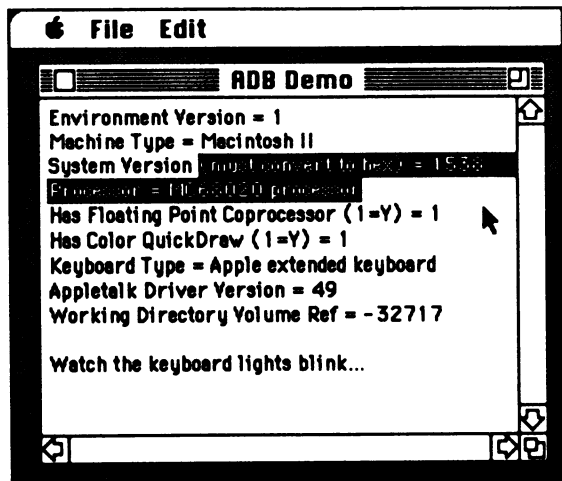


Fig. 6 Our Pascal Demo supports Text Editing

```
' SysEnvirons and ADB Demo
' ©1989 MacTutor
' by Dave Kelly
' ZBasic 5.0
'
```

```
'*****
' First check the System Environments
' To do this the SYSENVIRONS function needs to be
' modified to work. Replace old SYSENVIRONS with D0.W=FN
```

```
' SYSENVIRONS(D0.L, A0.L)
'*****
WINDOW OFF
COORDINATE WINDOW
'Find out screen size.
CALL GETWMGRPORT(WMgrPort&)
PortTop=PEEK WORD(WMgrPort&+8)
PortLeft=PEEK WORD(WMgrPort&+10)
PortBottom=PEEK WORD(WMgrPort&+12)
PortRight=PEEK WORD(WMgrPort&+14)

WINDOW 1,"Main Window", (10,44)-(PortRight-4,PortBottom-4),1
'Bring window to the front (necessary under Multifinder)
Wptr&=WINDOW(14)
CALL SELECTWINDOW(Wptr&)
' Set up SysEnvRec
DIM environsVersion,machineType,
systemversion,processor,hasFPU,keyBoardType,atDrvrvVersNum,sysVRefNum
OSErr%=FN SYSENVIRONS(1,VARPTR(environsVersion))
LONG IF OSErr%=0
PRINT "Environment Version: ";environsVersion
PRINT "Machine Type: ";
SELECT machineType
CASE 0
PRINT "new version of Macintosh"
CASE 1
PRINT "Macintosh 512K enhanced"
CASE 2
PRINT "Macintosh Plus"
CASE 3
PRINT "Macintosh SE"
CASE 4
PRINT "Macintosh II"
CASE -1
PRINT "Macintosh with 64K ROM"
CASE -2
PRINT "Macintosh XL"
END SELECT
PRINT "System
Version: ";LEFT$(HEX$(systemversion),2);". ";RIGHT$(HEX$(systemversion),2)
PRINT "Processor: ";
SELECT processor
CASE 0
PRINT "new processor"
CASE 1
PRINT "MC68000 processor"
CASE 2
PRINT "MC68010 processor"
CASE 3
PRINT "MC68020 processor"
END SELECT
b$="X"+RIGHT$(BIN$(hasFPU),8)
hasColorQD=VAL(b$)
b$="X"+LEFT$(BIN$(hasFPU),8)
hasFPU=VAL(b$)
PRINT "Has Floating Point Coprocessor: ";
IF hasFPU=1 THEN PRINT "Yes" ELSE PRINT "No"
PRINT "Has Color QuickDraw: ";
IF hasColorQD=1 THEN PRINT "Yes" ELSE PRINT "No"
PRINT "Keyboard Type: ";
SELECT keyBoardType
CASE 0
PRINT "Macintosh Plus keyboard with keypad"
CASE 1
PRINT "Macintosh keyboard"
CASE 2
PRINT "Macintosh keyboard and keypad"
CASE 3
PRINT "Macintosh Plus keyboard"
CASE 4
PRINT "Apple extended keyboard"
CASE 5
PRINT "standard Apple Desktop Bus keyboard"
CASE ELSE
```

```

PRINT "don't recognize this one!"
END SELECT
PRINT "AppleTalk Driver version: ";atDrvVrsNum
PRINT "Working Directory Volume Reference #:";sysVRefNum
ELSE
PRINT "Error =";OSErr%
END IF
'*****
' Now Read the ADB
' This routine turns on and off the
' lights of the Extended keyboard
' ADB calls and functions need to be added to ZBasic 5.0
' using the Toolbox mover program.
'*****

IF keyBoardType<>4 THEN END
DIM DeviceType%,ServiceAddress%,DataAddress%:' GETINDADB
Parameter block
DIM buffer%(2)
bufferptr%=VARPTR(buffer%(0))
compRoutptr%=0
Datablkptr%=0' ADBOP Parameter block

numberOfADBdevices=FN COUNTADBS
PRINT "There are";numberOfADBdevices;"ADB devices present."
IF numberOfADBdevices=0 THEN STOP
FOR i%=1 TO numberOfADBdevices
  ADBAdd%=FN GETINDADB(VARPTR(DeviceType%),i%)
  b%="&X"+RIGHT$(BIN$(DeviceType%),8)
  OrgADBAddress%=VAL(b%)
  b%="&X"+LEFT$(BIN$(DeviceType%),8)
  DeviceType%=VAL(b%)
  LONG IF ADBAdd%=2 'Got the address for the Extended
Keyboard
    Talk%=&H2E:' Talk command
    Listen%=&H2A:' Listen command
    Flush%=&H21:' Flush command
    PRINT "Press any key to continue..."
    DO
      X%=INKEY$
      GOSUB "MagicLights"
      UNTIL X%<>" "
    END IF
    NEXT i%
  LONG IF X%=""
    PRINT "Press any key to continue..."
    DO
      X%=INKEY$
      UNTIL X%<>" "
    END IF
  END IF
END

"MagicLights"
GOSUB "ToggleLightson"
GOSUB "Delay"
GOSUB "ToggleLightsoff"
GOSUB "Delay"
GOSUB "ToggleLightson"
GOSUB "Delay"
GOSUB "ToggleLightsoff"
GOSUB "Delay"
RETURN

"ToggleLightson"
compRoutptr%=ServiceAddress%:Datablkptr%=DataAddress%
'This call reads register 2 of the ext. keyboard.
OSErr%=FN ADBOP(VARPTR(bufferptr%),Talk%)
GOSUB "Delay"
buffer%(1)=&HF800
'The next call writes register 2 back to
'the extended keyboard. For some reason the register is not
'being written back to the keyboard. See Pascal version.
OSErr%=FN ADBOP(VARPTR(bufferptr%),Listen%)
RETURN

```

```

"ToggleLightsoff"
OSErr%=FN ADBOP(VARPTR(bufferptr%),Flush%)
RETURN

```

```

"Delay"
T1%=FN TICKCOUNT
DO
T2%=FN TICKCOUNT
UNTIL T2%-T1%=20
RETURN

```

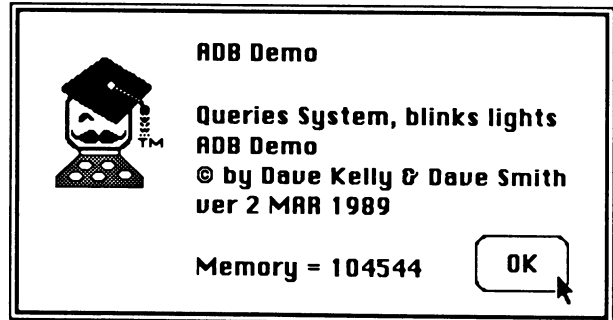


Fig. 7 Our Pascal About Box. Note Memory Function and ID String displayed from resources!

```

( ADBDemo )
( ©1989 MacTutor )
( By Dave Kelly, modified by David Smith )
( This program will blink the LEDs of the Extended Keyboard on
and off twice. This has no real value other than to show how
to communication with ADB devices. NOTE: It is not clear what
Apple has in mind for these LEDs, so you are using them at
your own risk. Refer to Technical Note #206 for more informa-
tion. Other tidbits include Multifinder friendly, about
dialog, vers resources, working sizable window. )

```

program ADBDemo;

uses

OSIntf, PrintTraps, MyADBGlobals, MyADBStuff;

procedure crash;

begin

ExitToShell;

end;

procedure InitMac;

begin

InitGraf(@thePort);

InitFonts;

InitWindows;

InitMenus;

TEInit;

InitDialogs(@crash);

InitCursor;

FlushEvents(everyEvent, 0);

Finished := false;

end;

procedure initRects;

var

MemoryPtr: ^Integer;

begin

MemoryPtr := pointer(mBarHeightGlobal);

mBarHeight := MemoryPtr;

screen := ScreenBits.Bounds; (current screen device)

SetRect(DragArea, Screen.left + 4, Screen.top + mBarHeight + 4, Screen.right - 4, Screen.bottom - 4);

SetRect(GrowArea, Screen.left + MinWidth, Screen.top +

```

MinHeight, Screen.right - 8, Screen.bottom - 8);
SetRect(ADBWindowRect, Screen.left + 15, Screen.top + 45,
Screen.right - 95, Screen.bottom - 95);
SetRect(ZoomRect, Screen.left + 4, Screen.top + mBarHeight +
4, Screen.right - 4, Screen.bottom - 4);
end;

```

```

procedure InitMyWindow;
var
    windtype: integer;
    Visible: boolean;
    GoAway: boolean;
    RefVal: LongInt;
    title: str255;
begin
    Visible := true;
    windtype := documentProc + ZoomBox;
    GoAway := true;
    RefVal := 0;
    title := 'ADB Demo';
    ADBWindow := NewWindow(nil, ADBWindowRect, title, Visible,
windtype, pointer(-1), GoAway, RefVal);
    ADBWindowPeek := WindowPeek(ADBWindow);
    SetPort(ADBWindow);
    TextFont(Geneva);
    TextSize(10);
    TextFace(1); (plain)
    TextMode(1); (Or)
    PenNormal;
    ForeColor(blackColor);
    BackColor(whiteColor);
    ADBWindowPeek.windowKind := userKind;
    with ADBWindow.portRect do
        begin
            SetRect(VCRect, right - (SBarWidth - 1), top - 1, right
+ 1, bottom - (SBarWidth - 2));
            SetRect(HCRect, left - 1, bottom - (SBarWidth - 1),
right - (SBarWidth - 2), bottom + 1);
            SetRect(GrowRect, HCRect.right, HCRect.top,
VCRect.right, HCRect.bottom);
            SetRect(ViewRect, left + 4, top + 4, right - (SBarWidth
- 1), bottom - (SBarWidth - 1));
        end; (of with)
        DestRect := ViewRect;
        myTextHandle := TNew(DestRect, ViewRect);
        title := '';
        VControl := NewControl(ADBWindow, VCRect, title, Visible,
0, 0, 0, ScrollBarProc, 1);
        ValidRect(VCRect);
        HControl := NewControl(ADBWindow, HCRect, title, Visible,
0, 0, 0, ScrollBarProc, 1);
        ValidRect(HCRect);
    end;

```

```

procedure InitMyPrint;
begin
    myPrint := THPrint(NewHandle(SIZEOF(TPrint)));
end;

```

```

procedure InitMyMenus;
var
    i: integer;
begin
    myMenus[AppleM] := GetMenu(AppleMenu);
    AddResMenu(myMenus[AppleM], 'DRVR');
    myMenus[FileM] := GetMenu(FileMenu);
    myMenus[EditM] := GetMenu(EditMenu);
    for i := 1 to MenuCount do
        InsertMenu(myMenus[i], 0);

```

```

    DisableItem(myMenus[FileM], fSave);
    DisableItem(myMenus[FileM], fSaveAs);
    DisableItem(myMenus[FileM], fPageSet);
    DisableItem(myMenus[FileM], fPrint);

```

```

    DisableItem(myMenus[EditM], eUndo);
    DrawMenuBar;
end; (of proc)

```

```

procedure doMouse (myEvent: EventRecord);
var
    whereIsIt: integer;
    whichWindow: WindowPtr;
    localPt, globalPt: Point;
    oldPort: GrafPtr;
begin
    globalPt := myEvent.where;
    localPt := globalPt; (global coord of mouse)
    GlobalToLocal(localPt); (local coord of mouse)
    whereIsIt := FindWindow(globalPt, whichWindow);
    case whereIsIt of
        inDesk: (0)
            doMessage('Mouse Click on Desktop.', '(Not handled in
this program.)', '', '');
        inMenuBar: (1)
            doMenuBar(MenuSelect(globalPt));
        inSysWindow: (2)
            SystemClick(myEvent, whichWindow);
        inContent: (3)
            doContent(myEvent, whichWindow);
        inDrag: (4)
            doDrag(whichWindow, globalPt);
        inGrow: (5)
            doGrow(whichWindow, globalPt, False);
        inGoAway: (6)
            if TrackGoAway(whichWindow, globalPt) then
                HideWindow(whichWindow);
        inZoomIn, inZoomOut: (7, 8)
            begin
                if TrackBox(whichWindow, globalPt, whereIsIt) then
                    begin
                        GetPort(OldPort);
                        SetPort(whichWindow); (safety device)
                        EraseRect(whichWindow.portRect);
                        ZoomWindow(whichWindow, whereIsIt, True);
                        doGrow(whichWindow, globalPt, True);
                        SetPort(OldPort);
                    end;
                end;
            otherwise
                begin
                    end;
            end; (of whereIsIt)
    end;

```

```

procedure doKeyDowns (myEvent: EventRecord);
var
    ch: char;
    charCode: longInt;
    keyCode: longInt;
begin
    charCode := BitAnd(myEvent.Message, charCodeMask); (strip
off key code)
    keyCode := BitShift(BitAnd(myEvent.Message, keyCodeMask),
-8); (strip off char)
    ch := Chr(charCode); (get keyboard char)
    if BitAnd(myEvent.Modifiers, CmdKey) = CmdKey then
        doMenuBar(MenuKey(ch)) (do menu command key)
    else
        begin (do keystroke)
            ParamText('No typing supported...', '', '', '');
            itemhit := CautionAlert(AlertDialog, nil);
            end; (of do key stroke)
    end; (of proc)

```

```

procedure doUpdates (myEvent: EventRecord);
var
    UpdateWindow: WindowPtr;
    TempPort: GrafPtr;

```

```

myRect: rect;
begin
  UpdateWindow := WindowPtr(myEvent.message);
  if UpdateWindow = ADBWindow then
    begin
      GetPort(TempPort); (save port)
      SetPort(UpdateWindow);
      BeginUpdate(UpdateWindow);
      EraseRect(UpdateWindow^.portRect);
      DrawGrowIcon(UpdateWindow);
      DrawControls(UpdateWindow);
      TEUpdate(UpdateWindow^.portRect, myTextHandle);
      EndUpdate(UpdateWindow);
      SetPort(TempPort); (restore port)
    end;
end; (of proc)

procedure doActivates (myEvent: EventRecord);
var
  TargetWindow: WindowPtr;
  TargetPeek: WindowPeek;
begin
  TargetWindow := WindowPtr(myEvent.message);
  TargetPeek := windowPeek(TargetWindow);
  SetPort(TargetWindow);
  if Odd(myEvent.modifiers) then
    begin (activate)
      if TargetWindow = ADBWindow then
        begin
          DisableItem(myMenus[EditM], eUndo);
          DrawGrowIcon(TargetWindow);
          TEActivate(myTextHandle);
          ShowControl(VControl);
          ShowControl(HControl);
        end
      end (of activate loop)
    else
      begin (deactivate)
        if TargetWindow = ADBWindow then
          begin
            EnableItem(myMenus[EditM], eUndo);
            EnableItem(myMenus[EditM], eCut);
            EnableItem(myMenus[EditM], eCopy);
            EnableItem(myMenus[EditM], ePaste);
            EnableItem(myMenus[EditM], eClear);
            DrawGrowIcon(TargetWindow);
            TEDeactivate(myTextHandle);
            HideControl(VControl);
            HideControl(HControl);
          end; (of my window activation)
        end; (of deactivate loop)
      end; (of proc)

procedure doMulti (myEvent: EventRecord);
const
  MouseMovedEvt = $FA;
var
  HiByte: byte;
  bit0: LongInt;
  sysresult: boolean;
  ResumeWindow: WindowPtr;
  ResumePeek: WindowPeek;
  SuspendWindow: WindowPtr;
  SuspendPeek: WindowPeek;
  MouseMove: LongAndByte;
begin
  bit0 := 31; (convert 68000 to toolbox)
  (check for mouse moved event)
  MouseMove.longView := myEvent.message;
  HiByte := Byte(MouseMove.byteView.byte0);
  if HiByte = MouseMovedEvt then
    begin (Handle mouse moved event)
      end;
    (check for resume event )

```

```

  if Odd(myEvent.message) then
    begin (resume)
      ( treat like activate Event if we are in front)
      ResumeWindow := FrontWindow;
      if ResumeWindow = ADBWindow then
        begin (adbWindow)
          SetPort(ResumeWindow);
          InvalRect(ResumeWindow^.portRect);
          ( force update event)
          DisableItem(myMenus[EditM], eUndo);
          ShowControl(VControl);
          ShowControl(HControl);
          DrawGrowIcon(ResumeWindow);
        end; (adbWindow)
      if FrontWindow <> nil then
        begin (DA check)
          ResumePeek := WindowPeek(FrontWindow);
          if ResumePeek^.windowKind < 0 then (DA)
            begin (da)
              myEvent.what := activateEvt;
              BitSet(@myEvent.modifiers, bit0);
              sysresult := SystemEvent(myEvent);
            end; (da)
          end; (DA check)
        ( end of activate Event)
      end (of resume)
    else
      begin (suspend)
        (de-activate Event)
        SuspendWindow := FrontWindow;
        if SuspendWindow = ADBWindow then
          begin (adbwindow)
            SetPort(SuspendWindow);
            InvalRect(SuspendWindow^.portRect);
            (force update)
            EnableItem(myMenus[EditM], eUndo);
            EnableItem(myMenus[EditM], eCut);
            EnableItem(myMenus[EditM], eCopy);
            EnableItem(myMenus[EditM], ePaste);
            EnableItem(myMenus[EditM], eClear);
            DrawGrowIcon(SuspendWindow);
            HideControl(VControl);
            HideControl(HControl);
          end; (adbwindow)
          if FrontWindow <> nil then
            begin (DA check)
              SuspendPeek := WindowPeek(FrontWindow);
              if SuspendPeek^.windowKind < 0 then
                begin (da)
                  myEvent.what := activateEvt;
                  BitClr(@myEvent.modifiers, bit0);
                  sysresult := SystemEvent(myEvent);
                end; (da)
              end; (DA check)
            ( end of de-activate Event)
          end; (suspend)
        end; (of proc)

procedure MainEventLoop;
const
  MultiFinderEvt = 15;
var
  sleep: LongInt;
  Event: EventRecord;
  DoIt: Boolean;
begin
  sleep := 10;
  repeat
    TEIdle(MyTextHandle);
    DoIt := WaitNextEvent(EveryEvent, Event, sleep, nil);
  (no mouse tracking)
  if DoIt then
    case Event.what of
      mouseDown:

```

```

        doMouse(Event);
    KeyDown, Autokey:
        doKeyDowns(Event);
    updateEvt:
        doUpdates(Event);
    activateEvt:
        doActivates(Event);
    MultiFinderEvt:
        doMulti(Event);
    otherwise
        begin
            end;
        end; (of event case)
until Finished; (end program)
end;

begin ( main )
    InitMac;
    InitRets;
    InitMyWindow;
    InitMyPrint;
    InitMyMenus;
    MainEventLoop;
end. ( end of main )

unit MyADBStuff;

interface

    uses
        OSIntf, PrintTraps, MyADBGlobals;

    procedure doMessage (message0: str255; message1: str255;
message2: str255; message3: str255);
    procedure doAbout;
    procedure doQuit;
    procedure doMenuubar (menuResult: LongInt);
    procedure doContent (ConEvent: EventRecord; contentWindow:
windowPtr);
    procedure doDrag (GrabWindow: WindowPtr; GlobalMouse:
point);
    procedure doGrow (ResizeWindow: WindowPtr; Globalmouse:
point; Zoomflg: Boolean);

implementation

    procedure doMessage; ((message0 : str255)
        (message1 : str255)
        (message2 : str255)
        (message3 : str255))

        var
            dialogP: DialogPtr;
            item: integer;
        begin
            ParamText(message0, message1, message2, message3);
            dialogP := GetNewDialog(MessageDialog, nil, pointer(-1));
            if dialogP = nil then
                begin
                    SysBeep(5);
                    ExitToShell;
                end;
            initCursor; (change to arrow)
            ModalDialog(nil, item);
            DisposDialog(dialogP);
        end;

    procedure delay;
        const
            stalltime = 15; (quarter seconds in ticks)
        var
            i: LongInt;
            tick1, tick2: LongInt;
        begin
            tick1 := TickCount;

```

```

            tick2 := tick1;
            repeat
                tick2 := TickCount;
            until tick2 >= tick1 + stalltime;
        end;

    procedure togglelightsoff;
        const
            flushcommandNum = $21; ( 00100001 )
        begin
            Buf.opServiceRtPtr := Info.dbServiceRtPtr;
            (service routine pointer)
            Buf.opDataAreaPtr := Info.dbDataAreaAddr;
            (optional data area address)
            OSEr := ADBOp(nil, nil, Buf.dataBuffPtr, flushcommandNum);
            ( Flush command, clears the device )
            if OSEr = -1 then
                doMessage('Unable to flush device', '', '', '');
            end;

    procedure togglelightson;
        var
            LEDaddress: ptr;
            LEDbyte: signedbyte;
        const
            talkcommandNum = $2E; ( 00101010 )
            listencommandNum = $2A; ( 00101110 )
        begin
            if OSEr = noErr then
                begin
                    Buf.opServiceRtPtr := Info.dbServiceRtPtr;
                    (service routine pointer)
                    Buf.opDataAreaPtr := Info.dbDataAreaAddr;
                    (optional data area address)
                    togglelightsoff;
                    OSEr := ADBOp(Buf.opDataAreaPtr, Buf.opServiceRtPtr,
Buf.dataBuffPtr, talkcommandNum);
                    if OSEr = -1 then
                        begin
                            doMessage('Unable to talk to device', '', '',
'');
                            exit(togglelightson);
                        end;
                    delay;
                    LEDaddress := POINTER(ORD(Buf.dataBuffPtr) + 2);
                    LEDbyte := LEDaddress^;
                    if LEDbyte = -1 then
                        LEDaddress^ := -8; (turn on)
                    OSEr := ADBOp(Buf.opDataAreaPtr, Buf.opServiceRtPtr,
Buf.dataBuffPtr, listencommandNum);
                    if OSEr = -1 then
                        begin
                            doMessage('Unable to listen to device', '', '',
'');
                            exit(togglelightson);
                        end;
                    end; (of noErr)
                end;
            end;

    procedure LightsMagic;
        begin
            togglelightson;
            delay;
            togglelightsoff;
            delay;
            togglelightson;
            delay;
            togglelightsoff;
            delay;
            togglelightson;
            delay;
            togglelightsoff;
            delay;
            togglelightson;

```

```

delay;
togglelightsoff;
delay;
end;

procedure QuerySystem;
var
    str1, str2: str255;
begin
    NumToString(LongInt(theWorld.environsVersion), str2);
    str1 := concat('Environment Version = ', str2, chr(13));
    TEInsert(pointer(ord4(@str1) + 1), Length(str1),
MyTextHandle);

    case theWorld.machineType of
        0:
            str2 := 'new version of Macintosh';
        1:
            str2 := 'Macintosh 512K enhanced';
        2:
            str2 := 'Macintosh Plus';
        3:
            str2 := 'Macintosh SE';
        4:
            str2 := 'Macintosh II';
        otherwise
            begin
                if theWorld.machineType = -1 then
                    str2 := 'Macintosh with 64K ROM';
                if theWorld.machineType = -2 then
                    str2 := 'Macintosh XL';
            end;
    end; (of case)
    str1 := concat('Machine Type = ', str2, chr(13));
    TEInsert(pointer(ord4(@str1) + 1), Length(str1),
MyTextHandle);

    NumToString(LongInt(theWorld.systemVersion), str2);
    str1 := concat('System Version (must convert to hex) = ',
str2, chr(13));
    TEInsert(pointer(ord4(@str1) + 1), Length(str1),
MyTextHandle);

    case theWorld.processor of
        0:
            str2 := 'new processor';
        1:
            str2 := 'MC68000 processor';
        2:
            str2 := 'MC68010 processor';
        3:
            str2 := 'MC68020 processor';
        4:
            str2 := 'MC68030 processor';
        otherwise
            begin
                str2 := 'unknown processor';
            end;
    end; (of case)
    str1 := concat('Processor = ', str2, chr(13));
    TEInsert(pointer(ord4(@str1) + 1), Length(str1),
MyTextHandle);

    NumToString(LongInt(theWorld.hasFPU), str2);
    str1 := concat('Has Floating Point Coprocessor (I=Y) = ',
str2, chr(13));
    TEInsert(pointer(ord4(@str1) + 1), Length(str1),
MyTextHandle);

    NumToString(LongInt(theWorld.hasColorQD), str2);
    str1 := concat('Has Color QuickDraw (I=Y) = ', str2,
chr(13));
    TEInsert(pointer(ord4(@str1) + 1), Length(str1),
MyTextHandle);

```

```

case theWorld.KeyboardType of
    0:
        str2 := 'Macintosh Plus keyboard with keypad';
    1:
        str2 := 'Macintosh keyboard';
    2:
        str2 := 'Macintosh keyboard and keypad';
    3:
        str2 := 'Macintosh Plus keyboard';
    4:
        str2 := 'Apple extended keyboard';
    5:
        str2 := 'Standard Apple Desktop Bus keyboard';
    otherwise
        begin
            str2 := 'Unknown keyboard value';
        end;
end; (of case)
str1 := concat('Keyboard Type = ', str2, chr(13));
TEInsert(pointer(ord4(@str1) + 1), Length(str1),
MyTextHandle);

    NumToString(LongInt(theWorld.atDrvrsVersNum), str2);
    str1 := concat('AppleTalk Driver Version = ', str2,
chr(13));
    TEInsert(pointer(ord4(@str1) + 1), Length(str1),
MyTextHandle);

    NumToString(LongInt(theWorld.sysVRefNum), str2);
    str1 := concat('Working Directory Volume Ref = ', str2,
chr(13));
    TEInsert(pointer(ord4(@str1) + 1), Length(str1),
MyTextHandle);
end;

procedure doADB;
var
    devTableIndex: integer;
    str1: str255;
begin
    TESSetSelect(0, TEMax, MyTextHandle);
    TESSetDelete(MyTextHandle);
    ShowWindow(ADBWindow);

    OSER := SysEnvirons(versRequested, theWorld);
    if OSER = envNotPresent then
        doMessage('System File older than 4.1!', '', '', '');
    if OSER = envBadVers then
        doMessage('Bad Version Requested', '', '', '');
    if OSER = envVersTooBig then
        doMessage('Requested Version Not Available', '', '',
'');
    if OSER = noErr then
        begin (Query and Blinking routine)

            QuerySystem;

            (toggle keyboard lights on and off)
            if theWorld.keyboardType = 4 then
                begin (keyboard type 4)
                    str1 := concat(chr(13), 'Watch the keyboard
lights blink... ', chr(13));
                    TEInsert(pointer(ord4(@str1) + 1), Length(str1),
MyTextHandle);

                    numberOfADBs := CountADBs;
                    for devTableIndex := 1 to numberOfADBs do
                        begin (devTableIndex)
                            ADBAddr := GetIndADB(info, devTableIndex);
                            ( find the device address )
                            if ADBAddr = 2 then
                                begin (ADBAddr)
                                    device := info.devType;

```

```

        LightsMagic;
        end; (of ADBAddr)
    end; (devTableIndex)
end; (keyboard type 4)
end; (of noErr)
end; (of Proc)

procedure doAbout;
var
    IDStrHandle: StringHandle;
    dialogP: DialogPtr;
    item: integer;
    Str1, Str2, Str3: str255;
    myHeapSpace: LongInt;
    FreeSpace: Size;
begin
    IDStrHandle := StringHandle(GetResource(rsrc, 0));
    if IDStrHandle = nil then
        begin
            doMessage('Get About box crash!', '', '', '');
            ExitToShell;
        end;
    MoveHi(Handle(IDStrHandle));
    HLock(Handle(IDStrHandle));
    FreeSpace := FreeMem;
    myHeapSpace := MaxMem(FreeSpace);
    NumToString(myHeapSpace, Str2);
    Str2 := concat('Memory = ', Str2);
    Str3 := '';
    Str1 := '';
    ParamText(IDStrHandle^, Str1, Str2, Str3);
    dialogP := GetNewDialog(AboutDialog, nil, pointer(-1));
    if dialogP = nil then
        begin
            doMessage('Dialog crash!', 'We are dead...', '', '');
            ExitToShell;
        end;
    initCursor;
    ModalDialog(nil, item);
    DisposDialog(dialogP);
    HUnlock(Handle(IDStrHandle));
end;

procedure doQuit;
begin
    DisposeWindow(ADBWindow);
    TEDispose(MyTextHandle);
    Finished := true;
end; (of proc)

procedure doSave;
begin
end;

procedure doSaveAs;
begin
end;

procedure doPrint;
begin
end;

procedure doPageSet;
begin
end;

procedure doMenubar; (menuResult: LongInt)
var
    theMenu: integer;
    theItem: integer;
    daName: STR255;
    accItem: integer;
    temp: GrafPtr;
    dummy: LongInt; (Desk Scrap result var)

```

```

    ScrapReturn: OSErr; (TEScrap result var)
    TextLength: integer;
    ScrapLength: LongInt;
begin
    theMenu := HiWord(menuResult); (menu)
    theItem := LoWord(menuResult); (item)
    case theMenu of
        AppleMenu:
            begin
                if theItem = aAbout then
                    doAbout
                else
                    begin (must be DA)
                        GetItem(myMenus[AppleM], theItem, daName);
                        GetPort(temp); (protect against flacky DA)
                        accItem := OpenDeskAcc(daName);
                        SetPort(temp);
                    end; (else)
                end; (of AppleMenu)
            FileMenu:
                begin
                    case theItem of
                        fADB:
                            begin
                                doADB;
                            end;
                        fSave:
                            begin
                                doSave;
                            end;
                        fSaveAs:
                            begin
                                doSaveAs;
                            end;
                        fPageSet:
                            begin
                                doPageSet;
                            end;
                        fPrint:
                            begin
                                doPrint;
                            end;
                        fQuit:
                            begin
                                doQuit;
                            end;
                        otherwise
                            begin
                                end;
                            end; (of theitem)
                    end; (of FileMenu)
            EditMenu:
                begin
                    if not SystemEdit(theitem - 1) then
                        begin
                            case theItem of
                                eUndo:
                                    begin
                                        doMessage('Undo not available.', '', '', '');
                                    end;
                                eCut:
                                    begin
                                        TECut(MyTextHandle);
                                        dummy := ZeroScrap;
                                        ScrapReturn := TEToScrap; (update desk scrap)
                                    end;
                                eCopy:
                                    begin
                                        TECopy(MyTextHandle);
                                        dummy := ZeroScrap;
                                        ScrapReturn := TEToScrap; (update desk scrap)
                                    end;
                                ePaste:
                                    begin

```

```

        ScrapReturn := TEFFromScrap;
        TextLength := MyTextHandle^.teLength;
        ScrapLength := TEGetScrapLen;
        if (LongInt(TextLength + ScrapLength)) >
longInt(TEMax - 1) then
    begin
        initCursor;
        paramText('Paste would exceed text
edit 32000 buffer limit!', '', '', '');
        ItemHit := StopAlert(AlertDialog, nil);
    end
    else
        TEPaste(MyTextHandle);
    end; (of paste)
eClear:
    begin
        TDelete(MyTextHandle);
    end;
    otherwise
    begin
        end;
    end; (of case)
    end; (of system edit)
    end; (of EditMenu)
    otherwise
    begin
        end;
    end; (of theMenu)
    HiliteMenu(0); (un-hilite selected menu)
end;

procedure doContent; ((ConEvent : EventRecord)
    (contentWindow : windowPtr));
var
    localPt, globalPt: Point;
    part: integer;
    myRect: Rect;
    control: ControlHandle;
begin
    if contentWindow <> FrontWindow then
        SelectWindow(contentWindow);
    globalPt := ConEvent.where;
    localPt := globalPt; (global coord of mouse)
    GlobalToLocal(localPt); (local coord of mouse)
    part := FindControl(localPt, contentWindow, control);

    if contentWindow = ADBWindow then
    begin
        SetPort(ADBWindow);
        if part <> 0 then
        begin (in control)
            end;
            if part = 0 then
            begin (content region)
                myRect := ADBWindow^.portRect;
                if PtInRect(localPt, myRect) then
                begin
                    TClick(localPt, BitAnd(ConEvent.modifiers,
ShiftKey) = ShiftKey, myTextHandle)
                end; (of ptInRect)
            end; (of part=0)
        end; (of contentwindow)
    end; (of proc)

    procedure doDrag; ((GrabWindow : WindowPtr)
        (GlobalMouse : point));
    begin
        DragWindow(GrabWindow, GlobalMouse, DragArea);
    end;

    procedure doGrow; ((ResizeWindow : WindowPtr);
        (GlobalMouse : point);
        (ZoomFlg: Boolean));
    var

```

```

        newSize: LongInt;
        hsize: integer;
        vsize: integer;
        oldPort: GrafPtr;
        myRect: rect;
        tempLong: LongInt;
        l, t, r, b: LongInt;
    begin
        if (ResizeWindow <> FrontWindow) then
            SelectWindow(ResizeWindow)
        else
            begin
                if (ZoomFlg) then
                begin
                    with ResizeWindow^.portRect do
                    begin
                        tempLong := bottom - top;
                        newSize := BitShift(tempLong, 16);
                        newSize := newSize + (right - left);
                    end;
                end
                else
                    newSize := GrowWindow(ResizeWindow, GlobalMouse,
GrowArea);
                if newSize <> 0 then
                begin (grow the window)
                    hsize := LoWord(newSize);
                    vsize := HiWord(newSize);
                    if ResizeWindow = ADBWindow then
                    begin
                        with ResizeWindow^.portRect do
                        (Pre-Grow)
                        begin
                            SetRect(VCRect, right - (SBarWidth - 1),
top - 1, right + 1, bottom - (SBarWidth - 2));
                            SetRect(HCRect, left - 1, bottom -
(SBarWidth - 1), right - (SBarWidth - 2), bottom + 1);
                            SetRect(GrowRect, HCRect.right,
HCRect.top, VCRect.right, HCRect.bottom);
                        end; (of with)
                        SizeWindow(ResizeWindow, hsize, vsize,
TRUE); (new portRect)
                        InvalRect(GrowRect);
                        EraseRect(GrowRect);
                        with ResizeWindow^.portRect do
                        (Post Grow)
                        begin
                            SetRect(VCRect, right - (SBarWidth - 1),
top - 1, right + 1, bottom - (SBarWidth - 2));
                            SetRect(HCRect, left - 1, bottom -
(SBarWidth - 1), right - (SBarWidth - 2), bottom + 1);
                            SetRect(GrowRect, HCRect.right,
HCRect.top, VCRect.right, HCRect.bottom);
                            SetRect(ViewRect, left + 4, top + 4, right
- (SBarWidth - 1), bottom - (SBarWidth - 1));
                        end; (of with)

                        InvalRect(GrowRect);
                        HideControl(VControl);
                        HideControl(HControl);
                        MoveControl(VControl, VCRect.left,
VCRect.top);
                        MoveControl(HControl, HCRect.left,
HCRect.top);
                        SizeControl(VControl, SBarWidth,
VCRect.bottom - VCRect.top);
                        SizeControl(HControl, HCRect.right -
HCRect.left, SBarWidth);
                        ShowControl(VControl);
                        ShowControl(HControl);
                        ValidRect(VCRect);
                        ValidRect(HCRect);

                        MyTextHandle^.ViewRect := ViewRect;

```



```

File

* Multifinder Quit name
Type mstr = STR
    ,101
Quit

Type FREF
    ,128
APPL 0
    ,129
TEXT 1

Type BNDL
    ,128
ADBx 0
ICN#
0 128 1 129
FREF
0 128 1 129

* — Multifinder events —

* bit 15 = reserved
* bit 14 = accept suspend resume events
* bit 13 = reserved
* bit 12 = can do background on null events
* bit 11 = multifinder aware
*         (activates & deactivates topmost
*         window at resume, suspend events)

Type SIZE = GNRL
    , -1
.H
4800 ;; $4800 = bits 14,11 set
.L
128000 ;; (for 150K recommended)
.L
800000 ;; (for 80K minimum)
.I

Type vers = GNRL
    , 1
.H
01;; byte vers # in BCD
10;; byte vers part 2 & 3
50;; byte release stage $50=release
00;; byte stage of non-release
00 00 ;; integer country 0=US
.P
V1.1 (US)
.P
ADB Demo 1.1, © by Dave Kelly & David Smith for MacTutor 1989

Type vers = GNRL
    , 2
.H
05;; byte vers # in BCD
30;; byte vers part 2 & 3
50;; byte release stage $50=release
00;; byte stage of non-release
00 00 ;; integer country 0=US
.P
V5.3
.P
MacTutor Volume 5 Number 3

* ——— menus ———

Type MENU
* the desk acc menu
    ,256
\14 ;;apple menu
    About ADB Demo...

(-

* the file menu
    ,257
File
    ADB /A
    (-
    (Save /S
    Save as...
    Page Setup... /U
    (Print... /O
    (-
    Quit /Q

* the edit menu
    ,258
Edit
    (Undo /Z
    (-
    Cut /X
    Copy /C
    Paste /V
    Clear

* — Dialogs —
* About Box dialog...
type DLOG
    ,256
About ADB Demo...
100 100 250 400
visible NoGoAway
1
0
256

type DITL
    ,256
3
BtnItem Enabled
112 235 141 284
OK

StatText Disabled
10 88 141 289
ADB Demo\0D\0D++
Queries System, blinks lights\0D^0\0D^1\0D^2\0D^3

PicItem Disabled
10 10 96 81
128

* Program Messages Dialog box...
type DLOG
    ,258
Program Messages
100 100 200 400
Visible NoGoAway
1
0
258

type DITL
    ,258
3
BtnItem Enabled
65 230 95 285
OK

StatText Disabled
15 60 85 222
^0\0D^1\0D^2\0D^3

IconItem Disabled

```

10 10 42 42

1

* — Alerts —

* Program error alerts...

type ALRT

,260

100 100 200 400

260

5555

type DITL

,260

2

BtnItem

65 230 95 285

OK

StatText Disabled

15 60 60 275

Program Problem Alert:\00*0*1*2*3

* misc resources

Type ICN* = GNRL

,128 (0)

.H

0001 0000 0002 8000 0004 4000 000B E000

0015 5000 0026 B800 0045 5400 0086 B200

0105 5100 0213 E080 0428 0040 0810 0020

1102 0010 2204 0008 4480 3F04 8910 4082

4220 8041 2441 3022 1081 C814 081E 7F8F

0402 3005 0201 0007 0100 8005 0080 6007

0040 1FE5 0020 021F 0010 0407 0008 8800

0004 1000 0002 2000 0001 4000 0000 8000

*

0001 0000 0003 8000 0007 C000 000F E000

001F F000 003F F800 007F FC00 00FF FE00

01FF FF00 03FF FF80 07FF FFC0 0FFF FFE0

1FFF FFF0 3FFF FFF8 7FFF FFFC FFFF FFFE

7FFF FFFF 3FFF FFFE 1FFF FFFC 0FFF FFFF

07FF FFFF 03FF FFFF 01FF FFFF 00FF FFFF

007F FFFF 003F FE1F 001F FC07 000F F800

0007 F000 0003 E000 0001 C000 0000 8000

Type ICN* = GNRL

,129 (0)

.H

0FFF FE00 0800 0300 0800 0280 0800 0240

0800 0220 0800 0210 0800 03F8 0800 0008

0800 0008 0801 0008 0802 8008 0805 4008

080A A008 0815 5008 082A A808 0815 5008

080A A008 0805 4008 0802 8008 0801 0008

0800 0008 0800 0008 0800 0008 0800 0008

0820 79E8 0850 4528 0888 45E8 09FC 4528

0A02 79E8 0800 0008 0800 0008 0FFF FFF8

*

0FFF FE00 0FFF FF00 0FFF FF80 0FFF FFC0

0FFF FFE0 0FFF FFF0 0FFF FFF8 0FFF FFF8

0FFF FFF8 0FFF FFF8 0FFF FFF8 0FFF FFF8

0FFF FFF8 0FFF FFF8 0FFF FFF8 0FFF FFF8

0FFF FFF8 0FFF FFF8 0FFF FFF8 0FFF FFF8

0FFF FFF8 0FFF FFF8 0FFF FFF8 0FFF FFF8

0FFF FFF8 0FFF FFF8 0FFF FFF8 0FFF FFF8

0FFF FFF8 0FFF FFF8 0FFF FFF8 0FFF FFF8

0FFF FFF8 0FFF FFF8 0FFF FFF8 0FFF FFF8

0FFF FFF8 0FFF FFF8 0FFF FFF8 0FFF FFF8

0FFF FFF8 0FFF FFF8 0FFF FFF8 0FFF FFF8

0FFF FFF8 0FFF FFF8 0FFF FFF8 0FFF FFF8

0FFF FFF8 0FFF FFF8 0FFF FFF8 0FFF FFF8

0FFF FFF8 0FFF FFF8 0FFF FFF8 0FFF FFF8

0FFF FFF8 0FFF FFF8 0FFF FFF8 0FFF FFF8

0FFF FFF8 0FFF FFF8 0FFF FFF8 0FFF FFF8

0FFF FFF8 0FFF FFF8 0FFF FFF8 0FFF FFF8

0FFF FFF8 0FFF FFF8 0FFF FFF8 0FFF FFF8

0FFF FFF8 0FFF FFF8 0FFF FFF8 0FFF FFF8

0FFF FFF8 0FFF FFF8 0FFF FFF8 0FFF FFF8

0FFF FFF8 0FFF FFF8 0FFF FFF8 0FFF FFF8

0FFF FFF8 0FFF FFF8 0FFF FFF8 0FFF FFF8

0FFF FFF8 0FFF FFF8 0FFF FFF8 0FFF FFF8

0FFF FFF8 0FFF FFF8 0FFF FFF8 0FFF FFF8

0FFF FFF8 0FFF FFF8 0FFF FFF8 0FFF FFF8

0FFF FFF8 0FFF FFF8 0FFF FFF8 0FFF FFF8

TYPE PICT = GNRL

,128

.I

891

195 254 281 325

.H

1101 A000 82A0 008E 0100 0A00 0000 0002

D002 4098 000A 00C3 00F8 00FF 0148 00C3

00FE 00FF 0145 00C3 00FE 00FF 0145 0000

02F7 0002 F700 02F7 0002 F700 02F7 0002

F700 02F7 0002 F700 02F7 0002 F700 02F7

0006 FD00 000E FC00 07FD 0001 1F80 FD00

07FD 0001 7FC0 FD00 07FD 0001 FFF0 FD00

08FE 0002 03FF FCFD 0008 FE00 0207 FFFE

FD00 09FE 0003 1FFF FF80 FE00 09FE 0003

3FFF FFE0 FE00 09FE 0003 7FFF FFF8 FE00

0A02 0000 01FE FF00 FCFE 0008 0200 0003

FDFF FE00 0A02 0000 0FFD FF00 C0FF 000B

0700 001F FFFF 3FFF E0FF 000B 0700 007F

FFFE 1FFF F8FF 000B 0700 00FF FFFE 1FFF

FCFF 000B 0100 01FE FF02 27FF FCFF 000B

0100 01FE FF02 F9FF F8FF 000B 0100 00FE

FF02 FE7F F0FF 000B 0200 003F FEFF 019F

E0FF 000B 0200 001F FEFF 01E7 C0FF 000B

0200 003F FEFF 01F9 80FF 000B 0200 0033

FEFF 01FE 80FF 000A 0200 0060 FDFF 00C0

FF00 0B07 0000 607F FFFF FCC0 FF00 0B07

0000 601F FFFF F870 FF00 0B07 0000 6007

FFFF F0F8 FF00 0B07 0000 6001 FFFF F0F8

FF00 0B07 0000 6000 FFFF F0F8 FF00 0B07

0000 6038 3FFF B050 FF00 0A06 0000 607C

0FFF 30FE 000B 0700 0060 F603 FE30 A8FF

000B 0700 0060 E301 FC30 50FF 000B 0700

0060 C000 7830 20FF 000B 0700 0060 0000

1030 88FF 000B 0200 0060 FE00 0130 50FF

000A 0200 0060 FE00 0030 FE00 0B02 0000

60FE 0001 30A8 FF00 0B07 0000 6807 0700

B050 FF00 0A06 0000 681F 8FC0 B0FE 000B

0700 006C 7FDF F1B0 A8FF 000A 0200 0067

FEFF 0030 FE00 0B09 0000 63FF FFFE 31F4

1000 0B09 0000 307F DFF0 6046 3000 0B09

0000 381F 8FC0 E045 5000 0B09 0000 1C00

0001 C044 9000 0B09 0000 0E00 0003 8044

1000 0802 0000 07FE FFFD 0009 0500 0001

FFFF FCFD 0008 FE00 0280 0004 FD00 9800

0A00 FF00 F801 1901 4800 FF00 FE01 1901

4500 FF00 FE01 1901 4500 0008 FE00 0280

0004 FD00 08FE 0002 FFFF FCFD 0008 0200

0001 FEAA FD00 0802 0000 03FE 55FD 000A

0600 0006 FEAFA800 FE00 0A06 0000 0D83

5835 40FE 000A 0600 001B 01B0 1AA0 FE00

0A06 0000 3501 5015 50FE 000A 0600 006A

82A8 2AA8 FE00 0A06 0000 D57D 57D7 F4FE

000A 0600 01AF AAFA AC1A FE00 0A06 0003

5055 5558 0DFE 000B 0700 06A0 2A02 A80A

80FF 000B 0700 0D60 3603 5415 40FF 000B

0700 0AB0 6B06 ABEA C0FF 000B 0700 0D5F

D5FD 5555 40FF 0009 0100 0AFC AA00 C0FF

0009 0100 0DFC 5500 40FF 0009 0100 0FFC

FF00 C0FF 0002 F700 02F7 0002 F700 02F7

0002 F700 02F7 0002 F700 A000 8FA0 0083

FF



Basic School

Pop Up Menus in True Basic



Dave Kelly
MacTutor Editorial Board
Volume 5, Number 4

New users of the Macintosh are sometimes fascinated with the use of pull-down menus. Even experienced users head straight for the menu bar when they get a new program to see just what it does. Usually the menus get checked out before the manual does. Hierarchical menus added additional capabilities for the user. A little over a year ago, we began seeing the appearance of pop-up menus as another interface for the user. The various versions of BASIC cover the regular pull-down menus pretty easily. Hierarchical menus and pop-up menus are now being used in most of the software which is now being released. This column will attempt to explain how to set up pop-up menus using True Basic. Similar methods for implementing may be used in ZBasic or any other language for that matter. The method is about the same in whatever language is used and since many of the commands come from the Macintosh Toolbox. It should be pointed out that you must be using System 4.1 or greater for Hierarchical or pop-up menus to work. You should upgrade to the latest version (6.02).

It is advisable to know when it is appropriate to use pop-up menus. Apple recommends that pop-up menus be used for setting values or choosing from lists of related items. A pop-up menu could "pop up" anywhere (its location is global), but usually used in a dialog. The indication that there is a pop-up menu is that there is a box with a one-pixel thick drop shadow which is drawn around the current value. When the user presses the mouse button which pointing within the box, the pop-up menu appears with the current value checked and highlighted. Other than that, the pop-up menu acts just like any other menu.

The Macintosh ROM takes care of the pop-up menu action, but you must take care of drawing the shadowed box which indicates that there is a pop-up menu. You must take care of inverting the title when the menu is showing. The current value should appear in the pop-up menu when it is selected. Apple also recommends that you NOT use Hierarchical pop-up menus. You should consider very carefully if you really need to use a pop-up menu. Some features could better be implemented with icons or with the normal menus in the menu bar. Don't use commands in your pop-up menus unless you have to. If you do use commands, you should duplicate the item in the menu bar menus. "All commands that are in general use throughout the application should appear in the menu bar. This assures that the most important commands are always visible and available to the user". (refer to "Commands in pop-up Menu" from HI Update #13).

How It works....

Inside Macintosh Volume. V menu manager routines added one new ROM function for doing pop-up menus. The function

PopUpMenuSelect allows pop-up menus to be handled and created anywhere on the screen. Since the pop-up menu resources are handled the same as for other menus, they may contain color and submenus. The True Basic demo program has a basic event loop structure which can be used as a basic skeleton for other applications if you want. The libraries which are used come from the True Basic Macintosh Programmer's Kit which is essential for any serious Macintosh program development.

To implement pop-up menus, the program uses an array to set up the menu items. These items are declared in the PopItem\$ array which is dimensioned near the beginning of the program. SysEnviron is called to see if the system has the pop-up menu routines. If it doesn't, the program quits. It would be better to display a dialog to warn the user why the program quit. In addition to setting up the menu bar, the menu used in the pop-up is created and stored in a resource where MyMenus\$(4) is the handle to the menu resource. After the menu resource is created, the only thing left to do is draw the pop-up title and shadowed box then wait for an event.

The top and left corner of the pop-up are assigned at the beginning of the Drawpopup routine so that you may place the pop-up wherever you like. Be sure to leave room for the title. The routine figures out the width of the Title and longest menu item and draws them at the appropriate locations. When the pop-up menu comes up, the shadowed box should always line up with the pop-up menu. This is simple, but you can get confused unless you realize that the shadowed box and title are drawn with local window coordinates, but the pop-up menu location is determined by global screen coordinates. It is necessary to convert this location from local to global to tell the pop-up select routine where the menu should go. If the window gets moved (by someone dragging it), remember that the global coordinate is now different, but the local coordinate stayed the same.

You determine that a pop-up menu needs to be handled by checking to see if the mouse was clicked within the shadowed box. The GetNextEvent function returns the point where the click occurred. This point is compared with the PtInRect routine to determine if the point is within the shadowed rectangle where the pop-up will appear. If so then the program handles the pop-up event by calling the PopUpMenuSelect routine. PopUpMenuSelect takes the menu handle, the left and top coordinates where the menu will appear, and the currently selected menu item and handles the menu by displaying the menu then returns the item number of the menu item that was selected. That item is then redrawn into the shadowed box and becomes the newly selected item. The program should then handle whatever action or flag the menu item should do.

This process is simple enough after seeing a few examples.

The main problem with pop-up menus as with other controls that you may want to use is that you must be careful not to clutter up the window too much. The graphic design of the interface is just as important as the code that is written. That's why there will always be a demand for good programs even though HyperCard and other tools make developing programs easier.

By the way, some of the programming in True Basic can be simplified by using the TrueWindows Library. The problem comes when you want to create an application using the Runtime package. The Binder utility asks for the compiled libraries and then the compiled program and outputs them to an application that contains the True Basic interpreter. If you use libraries that have calls or functions with the same name, they may work fine until you bind them together into an application with Binder. Binder gives an error when putting the libraries together which says that you have more than one call with the same name. This could happen if you are using TrueWindows and the other Macintosh Programmer's Kit routines (like the ones used in this program). You may be able to get around this by going to the source code files and removing the duplicate name and recompiling the library. Be sure to keep it separate from the original so that they don't get mixed up or someday you may need a routine that you deleted and wonder why things don't work.

I set up my copy of the True Basic runtime application with my own icon and when I create an application, the information is already set up provided that I want to use the same one each time I make a new application. Usually you will want them to be different. I use the Prof. Mac icon for any applications that I make for MacTutor.

```
! PopUp Menu Demo
! True Basic version 2.01
! Requires True Basic Macintosh Developer's Toolkit Libraries
! by Dave Kelly
! ©1989 MacTutor
```

```
REM Open up libraries
LIBRARY "MenuLib*"           ! Menu Manager
LIBRARY "WindowLib*"         ! Window Manager
LIBRARY "DeskLib*"           ! Desk Manager
LIBRARY "EventLib*"          ! Event Manager
LIBRARY "QuickLib*"          ! Quickdraw
LIBRARY "DataLib*"           ! Desk Acc and system calls
LIBRARY "MacLib*"            ! True Basic event control
LIBRARY "System*"            ! System Calls
```

```
REM following variables are used globally throughout program
DECLARE DEF NIL$,
POINTER$,screenBits$,bounds$,top,left,bottom,right,TopLeft$,H,V
DECLARE DEF OpenDeskAcc,NewWindow$,SystemEdit
DECLARE DEF MenuSelect,PtInRect,TrackGoAway
```

```
DIM MyMenus$(1:4)
DIM PopItem$(1:3)
CALL SysEnviron$(sysEnvRec$,status) ! Get current system
revision
CALL UnpackEnviron$(sysEnvRec$,envversion,
machine,sysversion,processor,
hasFPU,hasColorQD,keyboardtype,atversion, sysvrefnum)
IF sysversion=0 then ! Do we have the right ROM?
STOP
END IF
CALL TakeMac ! turn off True Basic and let the program do
its own thing
```

```
LET everyevent=-1 ! event mask for all events
LET doneFlag=0 ! this flag is set when
program ending has been selected.
LET z$=bounds$(screenBits$) ! Get the size of the
screen.
CALL setrect(r$,left(z$)+4,top(z$)+44,right(z$)-4,bottom(z$)-
4)
CALL setrect(dragrect$,4,24,right(z$)-4,bottom(z$)-4)
LET myWindow$=NewWindow$(NIL$,r$,"Sample",1,0,POINTER$(-
1),1,0)
! Create a window
CALL SetPort(myWindow$) ! Access the new window
CALL SetUpMenus ! Turn on menus
CALL Drawwindow ! Set up window info

! Main Event Loop

DO
CALL SystemTask ! Handle System tasks/DAS
CALL GetNextEvent(everyevent,theEvent$,eResult)
! check for events
IF eResult<>0 then ! if no event error occurred then...
CALL UnpackEvent(theEvent$,what,mess,when,where$,mod)
SELECT CASE what
! what represents the kind of event that occurred.
CASE 1 ! mouse down event occurred
CALL FindWindow(where$,whichWindow$,wResult)
SELECT CASE wResult
CASE 1 ! Event was in the menu bar
LET mResult=MenuSelect(where$)
CALL DoMenu(mResult)
CASE 2 ! Event was in a system window
CALL SystemClick(theEvent$,whichWindow$)
! Pass the event to the system
CASE 3 ! Event was in content region of a window
CALL GlobalToLocal(where$)
! convert coordinates for the window
IF PtInRect(where$,PopRect$)=1 THEN
! see if popup was selected
CALL PopUpEvent ! if so, handle the
popup event
END IF
CASE 4 ! Event in the window's drag region
CALL DragWindow(whichWindow$,where$,dragrect$)
CASE 6 ! Event in go-away region of active window
LET doneFlag=TrackGoAway(whichWindow$,where$)
CASE else
END SELECT
CASE 6 ! update event occurred
CALL Packb(w$,1,32,mess)
CALL BeginUpdate(w$)
CALL Drawwindow
CALL DrawPopUp
CALL EndUpdate(w$)
CASE else ! anything else?
END SELECT
END IF
LOOP until doneFlag<>0

CALL DisposeWindow(myWindow$) ! Throw away window handle
CALL ClearMenuBar ! Clear Menus
FOR i=Lbound(MyMenus$) to Ubound(MyMenus$)
CALL DisposeMenu(MyMenus$(i))
NEXT i
CALL GiveMac ! Return control back to
True Basic
STOP ! End the program

SUB DrawWindow ! Draw message in window
CALL textfont(2) ! Set font to New York font
CALL textsize(12) ! Set size to 12 point
CALL textface(1) ! Set text to bold
CALL textmode(0) ! Set to copy mode
CALL moveto(10,20)
```

```

CALL DrawString("True BASIC Version 2.0 PopUp Menu demo")
CALL textface(0)          ! Set text to plain
END SUB

SUB DoMenu(code)           ! handle Menu events
CALL Packb(s$,1,32,code)
LET MenuNumber=Unpackb(s$,1,-16)
LET MenuItem = Unpackb(s$,17,-16)
SELECT CASE MenuNumber
CASE 1                     ! Apple Menu
CALL GetItem(MyMenus$(1),MenuItem,name$)
LET mrefNum=OpenDeskAcc(name$)
CALL SetPort(mywindow$)
CASE 2                     ! File Menu
LET doneFlag=-1
CASE 3                     ! Edit Menu
LET z=SystemEdit(MenuItem+1)
CASE else
END SELECT
CALL HiliteMenu(0)
END SUB

SUB SetUpMenus
DECLARE DEF NewMenu$,StringWidth ! Declare variables used
DECLARE DEF GetFontInfo$         ! in toolbox functions

LET MyMenus$(1)=NewMenu$(1,chr$(20)) ! The first menu is
CALL AddResMenu(MyMenus$(1),"DRVR") ! Apple menu.
LET MyMenus$(2)=NewMenu$(2,"File") ! File menu is second
CALL AppendMenu(MyMenus$(2),"Quit")
LET MyMenus$(3)=NewMenu$(3,"Edit") ! Next the Edit menu
CALL AppendMenu(MyMenus$(3),"Cut")
CALL AppendMenu(MyMenus$(3),"Copy")
CALL AppendMenu(MyMenus$(3),"Paste")
LET PopTitle$="PopUp Menu Title: " ! Save pop up title
LET MyMenus$(4)=NewMenu$(4,PopTitle$) ! Create pop up menu
LET Popitem$(1)="Item 1"
LET Popitem$(2)="Item 2"
LET Popitem$(3)="Item 3"
LET NoOfPopItems=3
LET PopItem=1
FOR i=1 to 3
CALL AppendMenu(MyMenus$(4),Popitem$(i)) ! Add popup
items
NEXT i
FOR i=lbound(MyMenus$) to Ubound(MyMenus$)-1 ! put the
menus into
CALL insertMenu(MyMenus$(i),0) ! the menu bar
NEXT i
CALL InsertMenu(MyMenus$(4),-1) ! Add pop up menu
CALL CheckItem(MyMenus$(4),PopItem,1) ! check default
item

REM Get maximum length of PopUp items
CALL TextFont(0) ! Set to system font
CALL TextSize(12) ! Set to 12 point size
CALL GetFontInfo(FontInfo$)
LET ascent = Unpackb(fontinfo$,1,-16)
LET descent = Unpackb(fontinfo$,17,-16)
LET widMax = Unpackb(fontinfo$,33,-16)
LET leading = Unpackb(fontinfo$,49,-16)
LET MaxItemLength=0

FOR i=1 to NoOfPopItems
LET strwidth=StringWidth(Popitem$(i))
IF StrWidth>MaxItemLength then LET
MaxItemLength=StrWidth
NEXT i

CALL DrawPopUp
CALL DrawMenuBar
END SUB

SUB DrawPopUp

```

```

CALL TextFont(0) ! Set Font to Chicago (System)
CALL TextSize(12) ! Set Size to 12 point
LET Popuptop=100 ! Top of Popup menu
LET Popupleft=200 ! Left of Popup menu
CALL SetRect(PopRect$, Popupleft,Popuptop,
Popupleft+5*MaxItemLength+13, Popuptop+ascent+ descent+
leading+1)
CALL FrameRect(PopRect$) ! Draw currently selected item
CALL MoveTo(Right(PopRect$),Top(PopRect$)+1)
CALL LineTo(Right(PopRect$),Bottom(PopRect$))
CALL MoveTo(Left(PopRect$)+1,Bottom(PopRect$))
CALL LineTo(Right(PopRect$),Bottom(PopRect$))
LET StrWidth=StringWidth(PopRect$)
LET xlocation=Left(PopRect$)-StrWidth
LET ylocation=(Top(PopRect$)+Bottom(PopRect$))/2+(ascent-
descent)/2
CALL MoveTo(xlocation,ylocation)
CALL Drawstring(PopTitle$) ! Draw the Popup menu title
CALL SetRect(InvertTitleRect$,xlocation-
8,Top(PopRect$)+1,Left(PopRect$),Bottom(PopRect$))
LET xlocation=Left(PopRect$)+13
CALL MoveTo(xlocation,ylocation)
CALL Drawstring(PopItem$(PopItem))
! Draw the currently selected item
END SUB

```

```

SUB PopUpEvent
DECLARE DEF PopUpMenuSelect ! Declare function
CALL InvertRect(InvertTitleRect$) ! invert popup title
LET TempPoint$=TopLeft$(PopRect$)
CALL LocalToGlobal(TempPoint$) ! Change to global coords
LET PopTop=V(TempPoint$)+1
LET PopLeft=H(TempPoint$)+1
LET
Result=PopUpMenuSelect(MyMenus$(4),PopTop,PopLeft,PopItem)
! Do the Popup
CALL Packb(s$,1,32,Result) ! Get the menu result
LET MenuNumber=Unpackb(s$,1,-16)
! Ignore Menunumber, we know which menu this is
LET MenuItem = Unpackb(s$,17,-16) ! Get the menu item

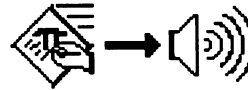
IF MenuItem=PopItem THEN
CALL InvertRect(InvertTitleRect$)
! Invert the title to normal if old item selected
ELSE
CALL CheckItem(MyMenus$(4),PopItem,0) ! uncheck last
item
CALL CheckItem(MyMenus$(4),MenuItem,1) ! check new item
CALL EraseRect(PopRect$) ! Draw the current menu item
CALL FrameRect(PopRect$)
CALL MoveTo(xlocation,ylocation)
CALL Drawstring(PopItem$(PopItem))
CALL InvertRect(InvertTitleRect$)
SELECT CASE MenuItem ! Handle menu event
CASE 1
REM Do Item 1
LET PopItem=MenuItem
CASE 2
REM Do Item 2
LET PopItem=MenuItem
CASE 3
REM Do Item 3
LET PopItem=MenuItem
CASE ELSE
END SELECT
CALL MoveTo(xlocation,ylocation)
CALL TextFont(0) ! Set font to Chicago (System)
CALL Drawstring(PopItem$(PopItem)) ! draw selected
pop up item
END IF
END SUB
END

```



Basic School

Sounding Off with MS QuickBasic



Ron Butcher
Libertyville, IL
Volume 5, Number 5

Playing Digitized Sounds with MS QuickBasic

[Ron Butcher bought his first computer in November of 1984; luckily, it was a Macintosh. He first learned BASIC on a DEC minicomputer at DeSoto, Inc., where he works as an Industrial Chemist. He discovered programming on the Mac with MS Basic 1.0 in December of 1984.]

There have been many occasions when I wished I could use digitized sounds instead of Basic's SOUND statements for enhancing programs I was writing. Judging from questions seen in magazines and on BBS's from other Basic programmers, I guess I wasn't alone. As I have a game written in Basic that could benefit greatly from sound effects, I decided to look into this further. This lead me to some heavy reading in Inside Macintosh.

Well yes, you can play digitized sounds from Basic, using the free form synthesizer in the Mac, and very handy Library routines implemented in QuickBasic. These routines allows you to implement Operating System ROM traps which return information in a Parameter Block. Many of these functions are used by the File Manager and Device Manager, such as _Open, _Close, _Read, _Write, etc. I suggest you beg or borrow (don't steal) a copy of Inside Mac so you can follow some of this background information.

Sound Background

To play digitized sounds, we need to utilize the Free-Form Synthesizer. Inside Mac's Pascal structure looks like this:

```
Type FFSynthRec = RECORD
    mode:      INTEGER; (always ffMode)
    count:     Fixed; ("sizing" factor)
    waveBytes: FreeWave(waveform description)
END;
```

```
FFSynthPtr = ^FFSynthRec;
```

```
FreeWave= PACKED ARRAY OF Byte;
```

Let's translate this for our use with QuickBasic. We need to allocate a block of memory for the size of the sound involved, and save the first 6 bytes of this block for the mode (integer fmode, which is always 0 for the free form synthesizer), and count (fixed point, which is how we tell the sound driver at what "speed" to play the sound). Inside Mac also tells us that we can play a free-form sound by making a Device Manager Write call with the following parameters: ioRefNum must be -4 (the Sound Driver), ioBuffer must point to the synthesizer buffer, and ioReqCount must contain the length of the synthesizer buffer. The sound data needs to be read into the buffer, the mode and count entered as the first six bytes, and that buffer (FFSynthRec) is then written to the Sound Driver.

The count constants, being fixed point, threw me a curve. When count is equal to 1, the "sizing factor" is 1, and sounds are played at the 22KHz sampling rate. But how to get .5 (11KHz), .33 (7.4KHz), and .25 (5.5KHz) presented a minor challenge. Fixed point numbers on the Mac are 32 bits long. The high word is the integer, and the low word the fractional part. The bit representation is shown in Fig. 1. For example, 1.0 would be represented as: 0000000100000000. 1.5 would be represented as: 0000000110000000. From this we get the count & hex values of &H10000 for 1.0, &H8000 for .5, &H5555 for .33, and &H4000 for .25. The actual sample rate is taken from this equation:

$$\text{frequency (Hz)} = 1000000 / (44.93 * (\text{wavelength} / \text{count}))$$

This is derived from the fact that the sound is created by starting play at the first byte, and skipping ahead count bytes every 44.93 microseconds; so with the smallest wavelength (1 byte) and a count value of 1, the frequency works out to 22,256.8 Hz, which is the highest quality sampling rate of most sound digitizing software.

Fixed Point Number Representation

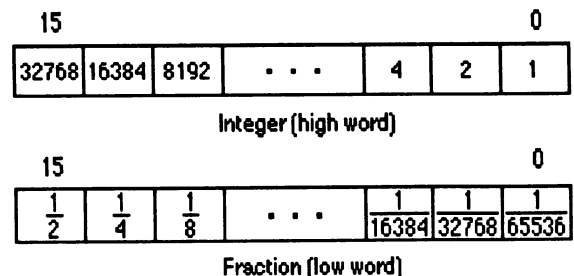


figure 1.

The ToolBox MBLC

The following illustrates the formats for the Operating System (OS) calls needed to be made to implement the Sound-Player program. The calls we are going to make are register based calls. As a general rule, OS calls are register based, and ToolBox calls are stack based, but there are exceptions on both sides (here I mean Mac ToolBox, not QuickBasic ToolBox calls). You can tell one Trap from another by checking bit 11 of the Trap word. If it is set (1), then it is a Toolbox Trap word; if bit 11 is not set (0), then the call is an OS Trap word. For example, traps starting with A0 are OS traps, and traps starting with A9 are Toolbox traps.

The generic register based QuickBasic call is:
ToolBox "R", Trap%, ReturnArray&(0), (a0&), (a1&), (d0&), (d1&), (d2&)

For our calls having values passed to and from a Parameter Block, the call in our program will look like this:

```
Param%=VARPTR(ParamBlock$(0))
```

```
ToolBox "R",Trap%,Reg$(0),(Param%)
```

Per Inside Macintosh, after we pass the appropriate values to the Parameter Block, we can make the call with a0 pointing to the address of the Block (Param%). Values are first passed to the appropriate registers (a0&.. etc) and ReturnArray Reg&() will contain the values passed to them after the call is made. Any errors will be returned in d0 (Reg&(2)). To get a pointer to our sound record, I've used NewPointer to allocate the blocksize:

```
NewPtr%=&HA01E
```

```
ToolBox "R",NewPtr%,Reg$(0),,(Siz%)
```

The size of the soundrecord is contained in Siz%, and the pointer to the record will be return in a0 (Reg&(0)). To manipulate the sound resources (snd), I've used a slightly different technique. I first use the QuickBasic ToolBox call GetHandleSize to obtain the size of the snd resource. NewPointer is then used to allocate the appropriate soundrecord. To coerce the snd resource Handle to a Pointer, I used BlockMove to transfer the sound information referenced by Handle h& to the block of memory pointed to by NewPointer. I could then release the memory used by the resource and the Handle using ReleaseRes. Note that in actuality, the header bytes of an snd resource contain resource info, including the length of the sound data (this length varies depending on the snd format; format 1 was developed for the Sound Manager, and format 2 is for HyperCard). I've ignored these bytes for simplicity's sake, but you can use them to your advantage in your own program. The program is set up to only open resource files of type SFIL and STAK, but you can expand on this to include other types. The same goes for the sound data files, which I have limited to type FSSD for this program example.

The File and Device Manager Routines

The following formats are for using File Manager _Open, _Close, and GetEOF calls. The —> indicates values to be passed to a routine, the <— indicates values returned by the routine, and <—> indicates values passed to and from the routine. The Parameter block size required is different for different calls, but the 80 byte array size used in the program is large enough for the calls we'll be implementing. The parameter block will be stored as an integer array.

```
FUNCTION PBOpen, PBClose, PBGetEOF (paramBlock: ParmBlkPtr;
async: BOOLEAN) : OSerr;
```

```
Trap_macro _Open, _Close, _GetEOF
```

```
(_Close only uses ioCompletion,ioResult, and ioRefNum)
```

```
(_GetEOF additionally uses ioMisc, which returns the logical end of file, or filelength)
```

```
Parameter block
```

->	12	ioCompletion	pointer
<-	16	ioResult	word
->	18	ioNamePtr	word
->	22	ioVRefNum	pointer
->	24	ioRefNum	long word
<-	26	ioVersNum	long word
->	27	ioPermssn	word
<->	28	ioMisc	long word

To open a sound data file, we need only to pass the address of the Pascal string of the file pathname (from FILE\$\$ and SADD function) to ioNamePtr, and call the _Open Trap. ioRefNum will be returned at byte 24 of the array. _GetEOF, which requires ioRefNum, can then be called to get the length of the file. You can also set byte 27 to 1 to make the Open call read only, just to be on the safe side. I used the QuickBasic B2PStr call to coerce the Pascal string from the Basic string. You could also use the Pstr\$=LEN(Bstr\$)+Bstr\$ method.

Here's the format for a File Manager _Read call and Device (or File) Manager _Write call:

```
FUNCTION PBWrite, PBRead (paramBlock: ParmBlkPtr; async:
BOOLEAN) : OSerr;
```

```
Trap_macro _Write, _Read
```

```
Parameter block
```

->	12	ioCompletion	pointer
<-	16	ioResult	word
->	24	ioRefNum	word
->	32	ioBuffer	pointer
->	36	ioReqCount	long word
<-	40	ioActCount	long word
->	44	ioPosMode	word
<->	46	ioPosOffset	long word

To read in all the sound file data into the buffer, we now need to give the Parameter block the address of the soundbuffer in ioBuffer, and how many bytes we want read in ioReqCount. Since all reads will be from the beginning of the file, ioPosMode and ioPosOffset are zero. ioRefnum is still the file reference number. When the Write call to the Sound Driver is made, ioRefNum is the Driver's reference number, -4 (&HFFFC). As we're set up here, the Write call is synchronous, that is, the program will not continue until the write call is finished. You can make an asynchronous call which allows your program to continue while the sound is being played by polling ioResult. This word returns a positive value while the sound is being played. This way you can limit certain operations of your program through a WHILE-WEND or IF-THEN operation. To make an asynchronous call, set bit 10 of the trap word; this would make the _Write trap &HA403, for example. I've found the asynchronous calls to be fairly touchy when being implemented from Basic. You can use _KillIO to stop any current sound being played and cancel every pending asynchronous call.

Bells and Whistles

The subroutine "Drawgraph" allows you to graphically illustrate the sound wave that has been opened for play. As byte values go, &H80 (128 decimal) represents 0 amplitude, while &HFF (255 decimal) represents 100% positive amplitude, and 0 represents 100% negative amplitude (See fig. 2). The routine takes a y axis sample every ScreenWidth\filelength bytes in order to have an even sampling over the width of the screen. Since the y values will be drawn down with respect to the screen coordinates, I implemented a simple inversion routine so the plot could be viewed right side up with respect to the actual wave. Although compressed, the graph that is drawn gives a decent visual

representation of the sound's actual waveform, such as that which might be seen on an oscilloscope in real time.

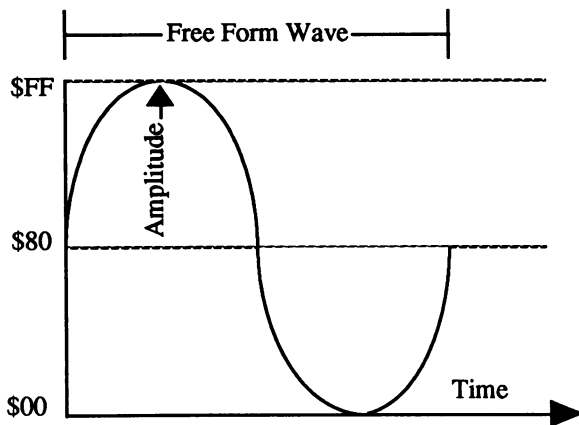


figure 2.

Thanks for the Memory

The main problem I ran into when setting up this program was finding an efficient way of allocating memory for the sound record, FFSynthRec. For a sound call, the memory block should ideally be locked until the sound call is finished, and then disposed of to free up the memory in the heap. I used one of QuickBasic's new features, the Toolbox calls, to use the NewPointer function to grab a non-relocatable block of memory to hold the sound record. In my initial version of the program, I used an array to hold the sound record. This works, but really slows things down a lot. The most efficient use of the array would be to dimension a dynamic array only to the size needed as dictated by size of the sound file and erase it after use. It is best to have a static array for the sound record, because some pretty weird things happen if that array shifts around in memory while being written to the Sound Driver. Ah, there's the rub.... you can't use static arrays in the interpreter, and if you use them in the compiler, you can't erase and resize them.

I've left a few items off of the program that may be implemented later, such as sound file decompression, and the ability to save sound resources as data, and visa-versa. Compressed sound data files have the header "HCOM", and I've flagged this condition with a BEEP sound, as I haven't figured out the decompression algorithm yet. Error handling is also at a minimum. For the Toolbox calls, Reg\$(2) returns the error code, which is 0 for no error. The OS errors returned can be many and varied, so you may want to show the error number in an Alert and stop the program. When compiling the program, check the "Make all arrays Static" and "Process Runtime Events" box. You can uncheck the boxes for "Use Default Menu" and "Use Default Window".

Being able to use the File Manager calls has other advantages which enable you to manipulate files in ways you can't do with normal BASIC commands such as INPUT, OUTPUT, WRITE, etc. Using the Toolbox calls, I've been able implement MacBinary in a telecom program, and write a disk cataloging utility, for example. Hopefully this program will shed some light

on using the sound driver from BASIC, and give you some ideas on how to use digitized sound in some of your own applications.

```
' SoundPlayer
' Plays digitized sounds from snd resources
' and sound data files using
' MicroSoft QuickBasic 1.0
' 2/26/1989
```

```
CLEAR, 100000%
DIM ParamBlock$(39),scr$(3),bar$(3),pt$(2),r$(3)
DIM PHS$(200),id$(200)
DIM a$(5)
' Trap calls
RITE$=&HA003:Open$=&HA000
Dread$=&HA002:Klose$=&HA001:GetEOF$=&HA011
NewPtr$=&HA01E : DisposePtr$=&HA01F
'Initialize variables and constants
FFsynthPtr$=0:Param$=0:Size$=0:h$=0:S$=0:ResH$=0
oldbut$=0:butnum$=0:Tag$=0:state$=0
x0$=0:y0$=0:y$=0:x$=0
true$=-1:false$=0:GotPointer$=false$:er$=0:
MenuItem$=0:MenuChoice$=0
ForkOpen$=false$:gotresource$=false$
num$=2:num1$=0:ref$=0:id$=0:action$=0
type$="snd":nam$="":F$="":G$="":pic$=""
lfd$=0:I$=0:count$=0:resid$=0
TwentyTwoK$=&H10000% ' count values
ElevenK$=&H8000%
SevenK$=&H5555%
FiveK$=&H4000%
ToolBox "i"
WINDOW CLOSE 1
MENU 1,0,1,"File"
MENU 1,1,1,"Open Sound Data"
MENU 1,2,1,"Open Sound Resource"
MENU 1,3,0,"Draw Sound Graph"
MENU 1,4,1,"Quit"
CmdKey 1,1,"D"
CmdKey 1,2,"R"
CmdKey 1,3,"G"
CmdKey 1,4,"Q"
Sheight=SYSTEM(6)
Swidh=SYSTEM(5)
WINDOW 2,"", (50,100)-(Swidh-32,Sheight-77),3
BUTTON 1,1,"22 KHz", (25,25)-(100,40),3
BUTTON 2,2,"11 KHz", (25,60)-(100,75),3
BUTTON 3,1,"7.4 KHz", (25,95)-(100,110),3
BUTTON 4,1,"5.5 KHz", (25,130)-(100,145),3
BUTTON 5,0,"Play", (130,75)-(190,100)
SetRect scr$(0),200,6,400,158
SetRect bar$(0),399,6,415,158
r$(0)=75
r$(1)=130
r$(2)=100
r$(3)=190
inSetRect r$(0),-4,-4
oldbut$=2:count$=ElevenK$
linenum$=0:top$=0:S$=0:in$=0
NewScroll S$,bar$(0),1,1,1,1
InactiveScroll S$
PICTURE ON
PENSIZE 3,3
FRAMEROUNDRECT VARPTR(r$(0)),16,16
PENNORMAL
ERASERECT VARPTR(scr$(0))
FRAMERECT VARPTR(scr$(0))
PICTURE OFF
pic$=PICTURE$
PICTURE ON : PICTURE OFF
refresh
ON MENU GOSUB MenSelect : MENU ON
```

```
ON Dialog GOSUB ChoiceWait : Dialog ON
```

```
UserWait:
WHILE true%
  hitTest
  ScrollText S%,scr$(0),PHS$(1),top%,num1%,linenum%,3
WEND
```

```
ChoiceWait:
MENU STOP : MOUSE STOP
action%=Dialog(0)
IF action%=5 THEN CALL refresh
IF action%<>1 THEN
  MENU ON : MOUSE ON
  RETURN
END IF
ON action% GOSUB HandleButton
MENU ON : MOUSE ON
RETURN
```

```
HandleButton:
butnum%=Dialog(1)
IF butnum%=oldbut% THEN RETURN
IF butnum%<>5 THEN
  BUTTON butnum%,2
  BUTTON oldbut%,1
  oldbut%=butnum%
END IF
SELECT CASE butnum%
CASE 1
  count%=TwentyTwoK&
CASE 2
  count%=ElevenK&
CASE 3
  count%=SevenK&
CASE 4
  count%=FiveK&
CASE 5
  GOSUB writefork
END SELECT
RETURN
```

```
openDataFile:
F$=FILES$(1,"FSSD"):IF F$="" THEN RETURN
BUTTON 5,0
gotresource% = false%
IF ForkOpen% THEN
  GOSUB CleanItUp
  ForkOpen%=false%
END IF
PtrTest
GOSUB ReadFork
Tag%=PEEK(FFsynthPtr+6) "HCOM"
IF Tag%=1212370765% THEN BEEP:RETURN 'compressed
BUTTON 5,1
RETURN
```

```
openResFile:
F$=FILES$(1,"SFILSTAK"):IF F$="" THEN RETURN
BUTTON 5,0
PtrTest
in%=0:top%=0:linenum%=0:resid%=0
GOSUB CountResources
GOSUB ClearBuffer
RETURN
```

```
Quit:
IF ForkOpen% THEN GOSUB CleanItUp
PtrTest
DisposeScroll S%
WINDOW CLOSE 2
END
```

```
MenSelect:
```

```
MenuItem%=MENU(0)
IF MenuItem%<>1 THEN RETURN
MenuChoice%=MENU(1)
MENU
ON MenuChoice% GOSUB openDataFile,openResFile,Drawgraph,Quit
RETURN
```

```
writefork:
IF gotresource% THEN GOSUB LoadResource
POKEW FFsynthPtr&,0 '0 for fmode
POKEW FFsynthPtr&+2,count% 'sizing value
Param%=VARPTR(Paramblock$(0))
POKE Param&+27,0 'reset permission
POKEW Param&+24,&HFFFC 'Sound Driver refnum
POKEW Param&+36,lfd% 'length to write
POKEW Param&+44,0 'ioPosMode
POKEW Param&+46,0 'ioPosOffSet
POKEW Param&+32,FFsynthPtr& 'address of synthrec
ToolBox "R",RITE%,Reg$(0),(Param%) 'call _Write
RETURN
```

```
ReadFork:
GOSUB ClearBuffer
B2PStr F$,G$
Param%=VARPTR(Paramblock$(0))
POKEW Param&+18,SADD(G$) 'address of filename
POKE Param&+27,1 'read only
ToolBox "R",Open%,Reg$(0),(Param%) 'call _Open
ToolBox "R",GetEOF%,Reg$(0),(Param%) 'call _GetEOF
Param%=VARPTR(Paramblock$(0))
lfd%=PEEK(Param&+28) 'file length
IF lfd%<0 THEN 'No data
  ToolBox "R",Klose%,Reg$(0),(Param%)
  BEEP : RETURN
END IF
Siz%=lfd%+6 '6 bytes for mode and count
GOSUB GetPointer
Param%=VARPTR(Paramblock$(0))
POKEW Param&+36,lfd% 'number of bytes to read
POKEW Param&+32,FFsynthPtr&+6 'block address
POKEW Param&+44,1 'ioPosMode read from start of file
ToolBox "R",Dread%,Reg$(0),(Param%) 'call _Read
ToolBox "R",Klose%,Reg$(0),(Param%) 'call _Close
RETURN
```

```
GetPointer:
'Get block of Siz& bytes and
'return address in FFsynthPtr&
ToolBox "R",NewPtr%,Reg$(0),,,(Siz&) 'call _NewPointer
GotPointer%=true%
FFsynthPtr&=Reg$(0)
RETURN
```

```
LoadResource:
IF resid%=id%(linenum%) THEN RETURN
PtrTest
'Get the resource
GetRes ref%,type$,id%(linenum%),h%
resid%=id%(linenum%)
HLock h%
GetHandleSize h%,lfd%
Siz%=lfd%
GOSUB GetPointer
'coerce pointer
ResH%=PEEK(h%)
BlockMove ResH%,FFsynthPtr&,Siz%
Hunlock h%
ReleaseRes h%
RETURN
```

```
CountResources:
IF ForkOpen% THEN GOSUB CleanItUp:ForkOpen%=false%
PtrTest
CountRes type$,num% 'number in System
```

```

ToolBox "WQ",&H997,F$,ref% 'openresfile
ForkOpen%=true%
updateResfile ref%
CountRes type$,num1%
num1%=num1%-num%      'number of type snd
GOSUB GetResources
RETURN

```

```

GetResources:
'set ResLoad to false
POKEW &HASE,0 'comment out for interpreter
FOR ind%=1 TO num1%
  GetIndRes type$,ind%,h&
  GetResInfo h&,id%(ind%),type$,nam$
  ReleaseRes h&
  PHS$(ind%)=nam$+" - "+STR$(id%(ind%))
  NEXT ind%
'set ResLoad to true
POKEW &HASE,-1 'comment out for interpreter
gotresource%=true%
ActiveScroll S&
RETURN

```

```

ClearBuffer:
Param%=VARPTR(Paramblock$(0))
FOR I=0 TO 79: POKE Param%+I,0: NEXT I
RETURN

```

```

CleanItUp:
CloseResfile ref%
InactiveScroll S&
in%=0:top%=0:linenum%=0:resid%=0:num1%=0
RETURN

```

```

Drawgraph:
DIALOG STOP :MENU OFF
WINDOW 3,"", (0,20)-(Swidth,Sheight),3
MOVETO 0,0
inc=lf&\Swidth
x0%=0:y0%=0:y%=0:x%=0
FOR I=1 TO lf& STEP inc
  LINETO x0%,y%
  y%=PEEK(FFsynthPtr&+(I))
  y%=(128-y%)+128 'Invert it
  PSET (x%,y%)
  x0%=x%
  x%=x%+1
NEXT I
TEXTFONT 0
MOVETO 180,300
DrawText "Click Mouse to Continue"
TEXTFONT 1
WHILE MOUSE(0)=0:WEND
WINDOW CLOSE 3

```

```

WINDOW 2
DIALOG ON:MENU ON
RETURN

```

```

SUB hittest STATIC
SHARED scr%(),pt%(),num1%,linenum%,gotresource%
in%=0
IF NOT gotresource% THEN EXIT SUB
IF MOUSE(0)=1 THEN
  GetMouse pt%(1)
  PtInRect pt%(1),scr%(0),in%
  IF in% THEN
    found
  END IF
  IF linenum%<>0 AND linenum%<=num1% THEN
    state%=1
  ELSE
    state%=0
  END IF
  BUTTON 5,state%
  refresh
END IF
END SUB

```

```

SUB found STATIC
SHARED linenum%,num1%,top%,pt%(),scr%()
linenum%=top%+(pt%(1)-scr%(0))\20
END SUB

```

```

SUB refresh STATIC
SHARED scr%(),PHS$(),S&,num1%,linenum%,pic$
PICTURE,pic$
top%=0
ScrollText S&,scr%(0),PHS$(1),top%,num1%,linenum%,3
END SUB

```

```

SUB PtrTest STATIC
SHARED GotPointer%,a&(),FFsynthPtr&
DisposePtr%=&HA01F
IF GotPointer% THEN 'call _DisposePointer
  CALL ToolBox("R",DisposePtr%,Reg&(0),(FFsynthPtr&))
  GotPointer%=0
END IF
END SUB

```

My thanks to Jim Reekes, whose Sound Manager document cleared up a lot of muddy mental audio, and to Michelle and all the helpful MicroSoft folks on the GENie MicroSoft Roundtable, who steered me in the right direction with some of the ToolBox syntax.



Basic School

Help for HELP Software



Dave Kelly
MacTutor Editorial Board
Volume 5, Number 6

In spite of the fact that Apple has referred to the Macintosh as the computer for "the rest of us", there are some people that still have trouble with computers of any kind, even the Macintosh. I think that Macintosh users sometimes look for complexity that is not there. So... there are many training courses, classes, etc. which may be taken to console those that may not feel comfortable with the application or system environment they are working with.

Many commercial application programs have developed help systems which allow the user to access on-line documentation of the application. The About... menu item (in the Apple menu) is typically the place to go to find some kind of help information. Users look for and expect consistency as they divide their time across several applications. This consistent atmosphere is why Macintosh users tend to use more different software applications than other computer users. On other systems users select a group of 3-4 applications and learn them well, but don't usually pick up other applications because of the time it takes to learn the new program's layout and how to get it to work. With the Macintosh, the consistent placement of the menus in the standard order (Apple Menu, File Menu, Edit Menu, etc.) helps to reduce the learning curve of new applications.

Even with this consistency, users sometimes don't have the manual handy or can't quite remember everything that the application does. This is the purpose of the on-line documentation. Help Software has recognized the need for a consistent help system. They have devised a system which developers may use to create on-line documentation quickly and easy for the user to use. Zedcor uses this help system to support on-line help for the ZBasic compiler. By typing HELP or accessing the About ZBasic™ dialog, the HELP DA is accessed and the ZBasic Help file is opened. Applications may access the HELP DA directly and request that specific messages be displayed in the HELP DA window. These are referred to as context sensitive or extended alerts by the HELP system.

I'm sure you can see the benefits of having an on-line documentation system as part of your application. For example, the user can type cmd-? or hit the help button (on the extended keyboard) and then click or select some object or control in the current application and the help system will provide a help message specific to the object selected. This is the type of system that MS Word and some other Microsoft products have implemented. But with HELP Software's DA, you don't have to write the on-line help system, it's already written for you.

It is the purpose of this column to introduce you to the HELP Software Desktop Help system and demonstrate a way to access context sensitive help messages from within your ZBasic application. Typically this involves more than just opening up the

Help DA. Under Multifinder, DAs load via the DA Handler unless the option key is held down when the DA is selected. For context sensitive help or extended alert messages, the application program needs to pass the message number to the Help DA so that the desired message can be displayed.

The LS Pascal code is divided into two projects. The Help project calls the Help DA with the option key pressed. The ContextHelp project calls the Help DA with the option key pressed and receives a integer message number which is passed from the calling application. Both projects are similar except for the message being passed. This code originated from the C example that Help Software provides with the Desktop Help system. The major difference is that the C code is a function instead of a procedure. I changed to a procedure because ZBasic doesn't provide an easy way to call a function. The CALL statement is meant for calling machine language or procedures from Pascal or C. Since it was tricky to get the return (error) value from the function, I decided to ignore the error and change to a procedure. The error flag would only be used to tell the user that the help system was not found or the help data file could not be loaded. An easier method may have been to rewrite the code to be executed directly in ZBasic. Due to time constraints I have not been able to try that yet; I'll leave that to you.

Since DAs don't load into the application space when Multifinder is active, the option key press is simulated. To do this the variable theKeyMap is used to pointed to the location \$178 where the key map offset to the option key is located. Modifier keys themselves do not generate keyboard events (see Inside Macintosh, I-246). The Help Software programmer followed the GetKey procedure with TMON and found the "master KeyMap" (theKeyMap). He decided to simulate an Option key press by putting there he same thing that pressing and holding the key would. The location of the option key at \$178 is not documented anywhere, but works on all ROMs from the 128K Mac thru the Plus, SE, II, IIX, and SE/30. If Apple doesn't change the key mapping then there shouldn't be any problem. One thing to remember is that if you press the option key this way, be sure to unpress the option key by changing theKeyMap back. The statement:

```
LongPtr(theKeyMap)^ := BitOr(LongPtr(theKeyMap)^, 4);  
presses the option key and:  
LongPtr(theKeyMap)^ := 0;  
releases the option key.
```

The help file is created with the Desktop Help system with the Help Editor. This file should be created after the application has been completely written, though may be updated and modi-

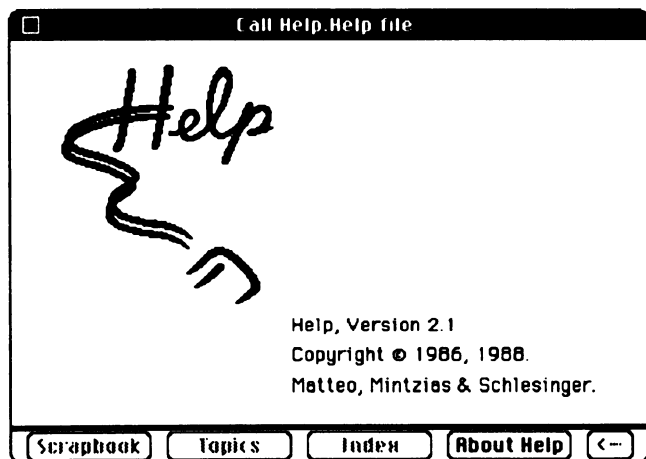
fied with the Help Editor throughout the entire development process. The file type of the help file should be set (with ResEdit) to the name of the creator of your application so that the help file which "belongs" to the application. DeskTop Help costs about \$395 and allows you to edit your own on-line documentation files for any or all applications you desire (even commercial ones). To distribute the Help DA you pay a yearly license of \$150 after the first year. The Help Editor should never be distributed, but can be purchased by anyone by buying the complete Desktop Help package.

The see how the context sensitive help works in the sample ZBasic program, press the cmd-? key to change the cursor to the "help mode". The program will then respond to menu selections and call a specific message corresponding to the message set up in the help file. The help key or the About... menu item will also select the help DA (with the option key pressed). If Multifinder is never used, the Help DA may be loaded directly without calling the Pascal procedure by using the OPENDESKACC function.

The ZBasic program loads the Pascal code into memory with the GETRESOURCE function. Next the code is locked in memory (so it won't move on us) and the ZBasic CALL statement executes the code. We can execute the code resource directly because LightSpeed Pascal puts a branch to the actual code at the beginning of the code header. After executing the code, the code is unlocked and the handle is released to free up memory.

These routines may be used with Pascal programs and BASIC without a lot of time and effort. The whole idea is to save time and provide a consistent way for the user to access on-line documentation or help.

Desktop Help is available from Help Software, 10659A Maplewood Road, Cupertino, Calif. 95014, tel. 408-257-3815. NOT COPY PROTECTED!



```
unit getHelp;
```

```
(Procedure for Calling the Help desk accessory )
(for Context Sensitive Help and Extended Alert Messages)
((Copyright ©1988 , Help Software , Inc . )
 ( Modified by Dave Kelly for MacTutor, April 1989 )
 ( Source code in Lightspeed Pascal 2.0 )
 ( This procedure will open the Help DA )
 ( Under MultiFinder , this procedure will install the Help DA
```

```
in * the )
( application heap . )
```

```
interface
```

```
procedure main;
```

```
implementation
```

```
procedure main;
```

```
type
```

```
LongPtr = ^LONGINT;
```

```
var
```

```
error, Help: integer; ( Help is the da refnum )
myhandle: Handle;
name: string;
theKeyMap: longint;
theKeyMapPtr: LongPtr;
```

```
begin
```

```
theKeyMap := $178;
```

```
Help := 0;
```

```
name := CONCAT(CHR(0), 'Help');
```

```
SetResLoad(FALSE); ( Don't load it, )
```

```
myhandle := GetNamedResource('DRVR', name);
```

```
( ...just get the handle )
```

```
error := ResError; ( -192 = Help not available )
```

```
SetResLoad(TRUE); ( Reset SetResLoad )
```

```
if (error = noErr) then
```

```
begin
```

```
EmptyHandle(myhandle); ( Try to purge the Help DA )
```

```
if myhandle = nil then
```

```
( If handle=NIL, it's not loaded, )
```

```
begin
```

```
ResrvMem(SizeResource(myhandle));
```

```
( ...reserve memory for it )
```

```
error := MemError; ( -108 = Not enough room in heap )
```

```
end;
```

```
if error = noErr then ( all go ... )
```

```
begin
```

```
LongPtr(theKeyMap)^ := BitOr(LongPtr(theKeyMap)^, 4);
```

```
( Required to work properly with MultiFinder )
```

```
( Press the Option key )
```

```
Help := OpenDeskAcc(name); ( Open the Help DA )
```

```
if (Help < 0) and (Help = (WindowPeek
(FrontWindow)^.windowKind)) then ( If the Help DA open )
```

```
begin
```

```
end ( end if Is the Help DA open )
```

```
else
```

```
begin
```

```
error := 1;
```

```
( Let the caller know that the Help DA was not opened )
```

```
LongPtr(theKeyMap)^ := 0;
```

```
( Release the Option key )
```

```
end;
```

```
end
```

```
else
```

```
begin
```

```
( Display an Alert : Not enough memory to open Help DA )
```

```
end
```

```
(Help is Available )
```

```
else
```

```
begin
```

```
( Display an Alert : Help DA not available )
```

```
end; ( End No Help Available )
```

```
end; (of Help procedure )
end.
```

```
unit getContextHelp;
```

```
(Procedure for Calling the Help desk accessory )
(for Context Sensitive Help and Extended Alert Messages)
((Copyright ©1988 , Help Software , Inc . )
 ( Modified by Dave Kelly for MacTutor, April 1989 )
 ( Source code in Lightspeed Pascal 2.0 )
```

Help.π		
Options	File (by build order)	Size
	DRVRRuntime.lib	0
	Interface.lib	0
D [N] V R	Help.p	0

(This procedure will open the Help DA and pass it the number)
 (of the Help message to be displayed .)
 (Under MultiFinder , this procedure will install the Help DA
 in * the application heap .)

interface

```
procedure main (message: integer);
  ( message: number of message to display )
```

implementation

```
procedure main (message: integer);
  type
    LongPtr = ^LONGINT;
  var
    error, Help: integer; (Help is desk accessory refnum )
    param: ParamBlockRec; (parameter blk for control call )
    myhandle: Handle;
    name: string;
    theKeyMap: longint;
    theKeyMapPtr: LongPtr;
  begin
    theKeyMap := $178;
    Help := 0;
    name := CONCAT(CHR(0), 'Help');
    SetResLoad(FALSE); ( Don't load it, )
    myhandle := GetNamedResource('DRVRR', name);
    error := ResError; ( -192 = Help not available )
    SetResLoad(TRUE); ( Reset SetResLoad )
    if (error = noErr) then
      begin
        ( Help is available )
        EmptyHandle(myhandle); ( Try to purge the Help DA )
        if myhandle = nil then
          ( If handle=NIL, it's not loaded, )
          begin
            ResrvMem(SizeResource(myhandle));
            ( ...reserve memory for it )
            error := MemError;
            ( -108 = Not enough room in heap )
          end;
        if error = noErr then ( all go ... )
          begin
            LongPtr(theKeyMap)^ := BitOr(LongPtr(theKeyMap)^, 4);
            ( Required to work properly with MultiFinder )
            ( Press the Option key )
            Help := OpenDeskAcc(name); ( Open the Help DA )
            if (Help < 0) and (Help =
(WindowPeek(FrontWindow)^.windowKind)) then
              ( If the Help DA open )
              begin
                param.ioCompletion := nil;
                param.ioRefNum := Help;
                ( Help is the value returned )
                ( by the OpenDeskAcc call )
                param.csCode := 5000;
                ( 5000 tells the Help DA: "This is )
                ( a context sensitive help call" )
                param.csParam[0] := message;
                ( Help message to display )
                error := PBCControl(@param, TRUE);
                ( asynchronous control call tells Help DA what to show ! )
              end
            end
          end
        else
          begin
```

```
          error := 1;
          ( Let the caller know that the Help DA was not opened )
          LongPtr(theKeyMap)^ := 0;
          ( Release the Option key )
        end;
      end
    else
      begin
        ( Display an Alert : Not enough memory to open Help DA )
      end
    end
  end
  ( Help is Available )
else
  begin
    ( Display an Alert : Help DA not available )
    end; ( End No Help Available )
  ( main:=error; for function )
end;
  (of GetContextHelp ( message ) procedure )
end.
```

ContextHelp.π		
Options	File (by build order)	Size
	DRVRRuntime.lib	0
	Interface.lib	0
D [N] V R	ContextHelp.p	0

'This program demonstrates the use of the
 'ZBasic 5.0 CALL statement when calling
 'a (LS) PASCAL CODE resource.

'©1989 MacTutor

'By Dave Kelly

```
WINDOW OFF
COORDINATE WINDOW
DEF MOUSE=-1
MOUSE ON
BREAK ON
DEFSTR LONG:T&=CVI("CODE")
DEF LEN=255
DIM B%(7)
GOSUB "Define Help Cursor"
APPLE MENU "About Call Help"
MENU 1,0,1,"File"
MENU 1,1,1,"Open..."
MENU 1,2,1,"Close"
MENU 1,3,0,"-"
MENU 1,4,1,"Save"
MENU 1,5,1,"Save As..."
MENU 1,6,0,"-"
MENU 1,7,1,"Page Setup..."
MENU 1,8,1,"Print..."
MENU 1,9,0,"-"
MENU 1,10,1,"Quit"
EDIT MENU 2
Message%0 'Set up string parameters
HelpCursor%0
```

```
'Find out screen size.
CALL GETWMGRPORT(WMgrPort&)
PortTop=PEEK WORD(WMgrPort&+8)
PortLeft=PEEK WORD(WMgrPort&+10)
PortBottom=PEEK WORD(WMgrPort&+12)
PortRight=PEEK WORD(WMgrPort&+14)
```

```
WINDOW 1,"Main Window",(-10,44)-(-PortRight-4,PortBottom-4),5
GET WINDOW #1, WindowPtr&
```

```
ON MENU GOSUB "Do MenuEvent"
```

```

ON DIALOG GOSUB "Do DialogEvent"
MENU ON:DIALOG ON:BREAK ON
"Loop"
CALL GETKEYS(B%(0))
LONG IF B%(2)=16 AND B%(3)=-32767
    GOSUB "ChangeCursor"
END IF
GOTO "Loop"

```

```

DIALOG OFF:MENU OFF

```

```

"Do MenuEvent"

```

```

Menunumber=MENU(0)

```

```

Menuitem=MENU(1)

```

```

MENU

```

```

SELECT Menunumber

```

```

CASE 255

```

```

    Message%=0

```

```

    GOSUB "Do Call"

```

```

CASE 1

```

```

    GOSUB "Do File Menu"

```

```

END SELECT

```

```

HelpCursor%=1

```

```

Message%=0

```

```

GOSUB "ChangeCursor"

```

```

RETURN

```

```

"Do File Menu"

```

```

LONG IF HelpCursor%=0

```

```

    SELECT Menuitem

```

```

        CASE 1 'Open

```

```

        CASE 2 'Close

```

```

        CASE 4 'Save

```

```

        CASE 5 'Save As...

```

```

        CASE 7 'Page Setup...

```

```

            DEF PAGE

```

```

        CASE 8 'Print...

```

```

            DEF LPRINT

```

```

        CASE 10 'Quit

```

```

            GOTO "Quit"

```

```

    END SELECT

```

```

XELSE

```

```

    SELECT Menuitem

```

```

        CASE 1 'Open

```

```

            Message%=1

```

```

            GOSUB "Do Call"

```

```

        CASE 2 'Close

```

```

            Message%=2

```

```

            GOSUB "Do Call"

```

```

        CASE 4 'Save

```

```

            Message%=4

```

```

            GOSUB "Do Call"

```

```

        CASE 5 'Save As...

```

```

            Message%=5

```

```

            GOSUB "Do Call"

```

```

        CASE 7 'Page Setup...

```

```

            Message%=7

```

```

            GOSUB "Do Call"

```

```

        CASE 8 'Print...

```

```

            Message%=8

```

```

            GOSUB "Do Call"

```

```

        CASE 10 'Quit

```

```

            Message%=10

```

```

            GOSUB "Do Call"

```

```

    END SELECT

```

```

END IF

```

```

RETURN

```

```

"Quit"

```

```

END

```

```

"Do DialogEvent"

```

```

D=DIALOG(0):DI=DIALOG(D)

```

```

SELECT D

```

```

CASE 1 'Button

```

```

CASE 2 'Edit Field

```

```

CASE 3 'Inactive Window

```

```

CASE 4 'Close Box

```

```

    GOTO "Quit"

```

```

CASE 5 'Window Update

```

```

CASE 6 'Return Key

```

```

CASE 7 'Tab Key

```

```

CASE 8 'Zoom-in

```

```

CASE 9 'Zoom-out

```

```

CASE 10 'Shift-Tab

```

```

CASE 11 'Clear Key

```

```

CASE 12 'Left-Arrow

```

```

CASE 13 'Right-Arrow

```

```

CASE 14 'Up-Arrow

```

```

CASE 15 'Down-Arrow

```

```

CASE 16 'Event inkey$

```

```

    IF DI=5 THEN GOSUB "Do Call"

```

```

CASE 17 'Disk Insert

```

```

END SELECT

```

```

RETURN

```

```

"Do Call"

```

```

LONG IF Message%=0

```

```

    Id=500 'Help DA

```

```

XELSE

```

```

    Id=501 'Context Sensitive Help

```

```

END IF

```

```

Result%&=0:Y%&=0

```

```

SCode%&= FN GETRESOURCE(T%&,Id) 'load the call into memory

```

```

IF SCode%&=0 THEN PRINT "Oops":END 'End if there is no call to
call

```

```

Y%&=FN HLOCK(SCode%&) ' Lock a Relocatable Memory Block

```

```

ActualCode%&=PEEK LONG (SCode%&)

```

```

LONG IF Message%=0

```

```

    CALL ActualCode%& 'Call the Help procedure

```

```

XELSE

```

```

    CALL ActualCode%&(Message%&) 'Call the Context Sensitive Help
function

```

```

END IF

```

```

CALL RELEASERESOURCE(SCode%&)

```

```

Y%&=FN HUNLOCK(SCode%&) 'Unlock a "Locked" Handle

```

```

Y%&=FN DISPOSHANDLE(SCode%&) 'Dispose of Memory block pointed
to by a 'Handle

```

```

Message%&=0 'Set up string parameters

```

```

RETURN

```

```

"Define Help Cursor"

```

```

DIM A$(40)

```

```

A$(0)=&H0000:A$(1)=&H3F80:A$(2)=&H7FC0:A$(3)=&H60C0

```

```

A$(4)=&H60C0:A$(5)=&H00C0:A$(6)=&H01C0:A$(7)=&H0380

```

```

A$(8)=&H0700:A$(9)=&H0E00:A$(10)=&H0C00:A$(11)=&H0C00

```

```

A$(12)=&H0000:A$(13)=&H0C00:A$(14)=&H0C00:A$(15)=&H0000

```

```

A$(16)=&H3F80:A$(17)=&H7FC0:A$(18)=&HFFE0:A$(19)=&HFFE0

```

```

A$(20)=&H1E00:A$(21)=&H1E00:A$(22)=&H03E0:A$(23)=&H07C0

```

```

A$(24)=&H0F80:A$(25)=&H1F00:A$(26)=&H1E00:A$(27)=&H1E00

```

```

A$(28)=&H1E00:A$(29)=&H1E00:A$(30)=&H1E00:A$(31)=&H1E00

```

```

A$(32)=&H0004:A$(33)=&H0002

```

```

HelpCursor%=0

```

```

RETURN

```

```

"ChangeCursor"

```

```

LONG IF HelpCursor%=0

```

```

    CALL SETCURSOR(A$(0))

```

```

    CALL SHOWCURSOR

```

```

    HelpCursor%=1

```

```

XELSE

```

```

    CURSOR=0

```

```

    HelpCursor%=0

```

```

END IF

```

```

WHILE B%(2)=16 AND B%(3)=-32767

```

```

    CALL GETKEYS(B%(0))

```

```

WEND

```

```

RETURN

```



Basic School

Useful Routines For ZBasic



Dave Kelly
MacTutor Editorial Board
Volume 5, Number 7

Hello! Are any of you tired of switching back and forth between versions of Basic? Although ZBasic still holds an edge over other Basics in the competition, there are benefits in favor of using other Basics from time to time. The frustrating thing about it is when you try to use a command, syntax, or subroutine with the wrong version of Basic. It's easy to do if you switch back and forth very often. It's also frustrating when your favorite function isn't implemented the same (or not at all) from what you are used to.

Charles Stricklin has sent in three functions which he wished were already implemented in ZBasic. These have been designed to add to your ZBasic library of routines. The GETIND-STRING\$ function works similarly to the GETINDRESOURCE function in the Macintosh toolbox. This routine allows easy retrieval of strings stored in a 'STR#' resource.

The GETRESWINDOW and GETRESMENU functions will get WIND and MENU resource data and interpret the resource to open up windows and setup menus using standard ZBasic statements. This is useful if you would like to use resources like 'real' applications do.

The program demonstrates the use of these functions. In developing this and through debugging the functions I am now even more aware of the missing capability to include resources from a resource file when compiling with ZBasic. This is a deficiency which I would hope Zedcor has as #1 on their 'TODO' list. Charles also listed this problem as high priority and wrote a short routine as a work around for the meantime:

debugging = -1: 'Or TRUE if defined previously)TRUE during development.

```
"OpenResourceFork"      : 'used in development only
LONG IF debugging       : 'we're still writing this
CALL PARAMTEXT("Do you want to use a specific
resource", "file?", "", "")
myAnswer = FN CAUTIONALERT(3, NIL): 'gets my answer from
alert
LONG IF myAnswer = 1    : 'I said "YES"
resourceFilename$ = FILES$(1, "rsrc", volume) : 'gets a
filename
                                '(created with ResEdit)
refNum = FN OPENRESFILE(resourceFilename$) : 'and
opens it's
'resource fork
END IF : '(myAnswer = 1)
END IF : '(debugging)
RETURN : '("OpenResourceFork")
```

Other items on the ZBasic "Wish List" include:
(to be fixed):

- The "Save Changes?" alert should only be called after the user has actually made changes to an existing document... especially not right after launching ZBasic itself when opening an existing document.
- ZBasic doesn't run reliably under MultiFinder. After several passes editing and running, ZBasic aborts even with 2.5+ Meg memory allocated to ZBasic. I've never had this happen when not running Multifinder. I suggest that Multifinder be turned off when editing and debugging ZBasic programs.

(to be improved):

- The 'Find' command is somewhat useless without a 'Change' command.
- The editor, even with the vast improvements over earlier versions, is still clumsy in several (many) areas.
- The default tab settings are a nuisance. This should be user definable.
- Editing would be much easier if more than one edit window (different files too) could be open at the same time.
- Lower the price. The ToolBox Editor should have been part of the package to begin with. The Program Generator is useful if you don't already have your own custom BASIC shell defined.

On the plus side, ZBasic is still the fastest and has been very reliable since version 5.0 was released. If the price were still under \$100, it would be a bargain. At the present time Light-Speed Pascal or C is a much better bargain when compared with ZBasic.

I'm sure that there are others routines which MacTutor readers have been using which would be useful to others. Send your routines to me via MacTutor and see your name in print! Thanks Charles!

```
' This program contains 3 new subroutines which
' may be used with the ZBasic Program Generator.
' ©1989 MacTutor
' Subroutines by Charles Stricklin, modified by Dave Kelly
' *****
'               o CONFIGURATION o
' *****
'
'           Default Variable Type: Integer
'           Convert to Upper Case: Yes
'           Space Req. After Key Words: Yes
'           Array Base 1: No
'           Bundle Bit: Yes
```



```

'
'*****
'          o SET UP VARIABLES o
'*****
WINDOW OFF:COORDINATE WINDOW:DEF MOUSE=-1:WIDTH -2
DIM T,L,B,R,MY,MX'          REQUIRED FOR TOOL BOX CALLS
'
WND=1:GOSUB"BUILD WINDOW"
GOSUB "BUILD MENU"
GOTO"EVENT QUEUE"
'*****
'          o THE QUEUE o
'*****
"EVENT QUEUE"
ON DIALOG GOSUB "HANDLE DIALOG":DIALOG ON
ON BREAK GOSUB "HANDLE BREAK":BREAK ON
ON MOUSE GOSUB "HANDLE MOUSE":MOUSE ON
ON MENU GOSUB "Handle Menu":MENU ON
:
"LOOP"
GOTO"LOOP"
:
DIALOG OFF:BREAK OFF:MOUSE OFF
'*****
'          o DIALOG ROUTINES o
'*****
"HANDLE DIALOG"
:
ACT=DIALOG(0):REF=DIALOG(ACT)
:
IF ACT=3 THEN WINDOW REF:RETURN
IF ACT=4 THEN GOSUB"CAPTURE":WINDOW CLOSE REF:END
IF ACT=5 THEN "FORMAT WINDOW"
IF ACT=11 THEN EDIT FIELD REF,""
:
LONG IF WINDOW(0)=1
END IF
:
RETURN
'*****
'          o BREAK ROUTINE o
'*****
"HANDLE BREAK"
END
'*****
'          o MOUSE ROUTINES o
'*****
"HANDLE MOUSE"
:
MACT=MOUSE(0):MX=MOUSE(1):MY=MOUSE(2)
:
RETURN
"Handle Menu"
MenuNumber=MENU(0)
Menuitem=MENU(1)
MENU
IF MenuNumber=1 AND Menuitem=12 THEN END
RETURN

'GETINDSTRING$ function By Charles Stricklin
'Modified by Dave Kelly for MacTutor, May, 1989

'This function is identical to the procedure GetIndString
'which is not in ROM and not supported by ZBasic. It reads
'a string from a string list and returns a copy of it in the
'variable the String$. If the string list doesn't exist of
the
'index is out of range an empty string is returned.

' stringListID is the resource ID of the string list; it's
resource type is 'STR'.
' stringIndex is the index of the desired string within the
list.

```

```

LONG FN GETINDSTRING$(StringListID, StringIndex)
TheString$=""
Offset=0
FALSE=0
MyHandle%=FN GETRESOURCE(CVI("STR"),StringListID)
LONG IF FN RESERROR=FALSE
MyPointer%=USR 3(MyHandle%)
NumberOfStrings=PEEK WORD(MyPointer%)
LONG IF (StringIndex>0) AND (StringIndex <= Number-
OfStrings)
LONG IF StringIndex>1
FOR ThisString=1 TO StringIndex-1
LengthOfThisString=PEEK(MyPointer%+2+Offset)
Offset=Offset+LengthOfThisString+1
NEXT
END IF
LengthOfDesiredString=PEEK(MyPointer%+2+Offset)
FOR Character=1 TO LengthOfDesiredString
TheString$=TheString$+CHR$(PEEK(MyPointer%+2+Offset+Character))
NEXT
END IF
MyHandle%=USR 7(MyPointer%)
CALL DETACHRESOURCE(MyHandle%)
END IF
END FN= TheString$

'GETRESMENU function By Charles Stricklin
'Modified by Dave Kelly for MacTutor, May, 1989
'This function creates a ZBasic menu from a MENU resource.
'The result given is boolean; true if the task is accom-
plished,
'false if there's a problem (the resource doesn't exist, etc.)

' resourceID is resource ID of the menu resource to be used.
' menuNumber is the ZBasic number of the menu to be created.

LONG FN GETRESMENU(ResourceID, MenuNumber)
DEFSTR LONG
Done=0
FALSE=0
MenuBlkSize=14
Title$=""
MyHandle%=FN GETRESOURCE(CVI("MENU"),ResourceID)
LONG IF FN RESERROR=FALSE
MyPointer%=USR 3(MyHandle%)
EnableFlags%=PEEK LONG(MyPointer%+10)
State=VAL(MID$(BIN$(EnableFlags%),32,1))
LengthTitle=PEEK(MyPointer%+MenuBlkSize)
FOR Character=1 TO LengthTitle
Title$=Title$+CHR$(PEEK(MyPointer%+MenuBlkSize+Character))
NEXT
MENU MenuNumber,0,State,Title$
Offset%=MenuBlkSize+LengthTitle+1
DO
Menuitem=Menuitem+1:Title$=""
LengthTitle=PEEK(MyPointer%+Offset%)
FOR Character=1 TO LengthTitle
Title$=Title$+CHR$(PEEK(MyPointer%+Offset%+Character))
NEXT
Offset%=Offset%+LengthTitle+1
IconNumber%=PEEK(MyPointer%+Offset%)
IF IconNumber% THEN
Title$=Title$+"~"+RIGHT$(STR$(IconNumber%),1)
CommandKey=PEEK(MyPointer%+Offset%+1)
IF CommandKey THEN Title$=Title$+"/"
"+CHR$(CommandKey)
MarkChar = PEEK(MyPointer%+Offset%+2)
IF MarkChar THEN Title$=Title$+"!"+CHR$(MarkChar)
CharacterStyle=PEEK(MyPointer%+Offset%+3)
IF FN BITTST(CharacterStyle,0) THEN
Title$=Title$+"<B"
IF FN BITTST(CharacterStyle,1) THEN

```

```

Title$=Title$+"<I"
IF FN BITTST(CharacterStyle,2) THEN
Title$=Title$+"<U"
IF FN BITTST(CharacterStyle,3) THEN
Title$=Title$+"<O"
IF FN BITTST(CharacterStyle,4) THEN
Title$=Title$+"<S"
State=VAL(MID$(BIN$(EnableFlags&),32-MenuItem,1))
Offset&=Offset&+4
MENU MenuNumber, MenuItem, State, Title$
UNTIL PEEK(MyPointer&+Offset&)=FALSE
Done=-1
END IF
MyHandle&=USR 7(MyPointer&)
CALL DETACHRESOURCE(MyHandle&)
END FN=Done

'GETRESWINDOW function By Charles Stricklin
'Modified by Dave Kelly for MacTutor, May, 1989
'This function creates a ZBasic window from a window template
'resource. Result given is boolean;true if task is accom-
plished,
'false if there's a problem ( resource doesn't exist, etc.)
'resourceID is the resource ID of window template to be used.
' windowNumber is ZBasic number of the window to be created.
' modal is boolean.

LONG FN GETRESWINDOW(ResourceID, WindowNumber, Modal)
Done=0
Title$=""
FALSE=0
MyHandle&=FN GETRESOURCE(CVI("WIND"),ResourceID)
LONG IF FN RESERROR=FALSE
MyPointer&=USR 3(MyHandle&)
Y1=PEEK WORD(MyPointer&)
X1=PEEK WORD(MyPointer&+2)
Y2=PEEK WORD(MyPointer&+4)
X2=PEEK WORD(MyPointer&+6)
Type=PEEK WORD(MyPointer&+8)+1
HasClose=PEEK WORD(MyPointer&+12)
LengthOfTitle=PEEK(MyPointer&+18)
FOR Character=1 TO LengthOfTitle
Title$=Title$+CHR$(PEEK(MyPointer&+18+Character))
NEXT
IF Type=13 THEN Type=9: 'ZBasic doesn't support
ZoomNoGrow
IF HasClose=FALSE THEN Type=Type+256
IF Modal THEN Type=-Type
WINDOW WindowNumber,Title$, (X1,Y1)-(X2,Y2),Type
Done=-1
END IF
MyHandle&=USR 7(MyPointer&)
CALL DETACHRESOURCE(MyHandle&)
END FN=Done

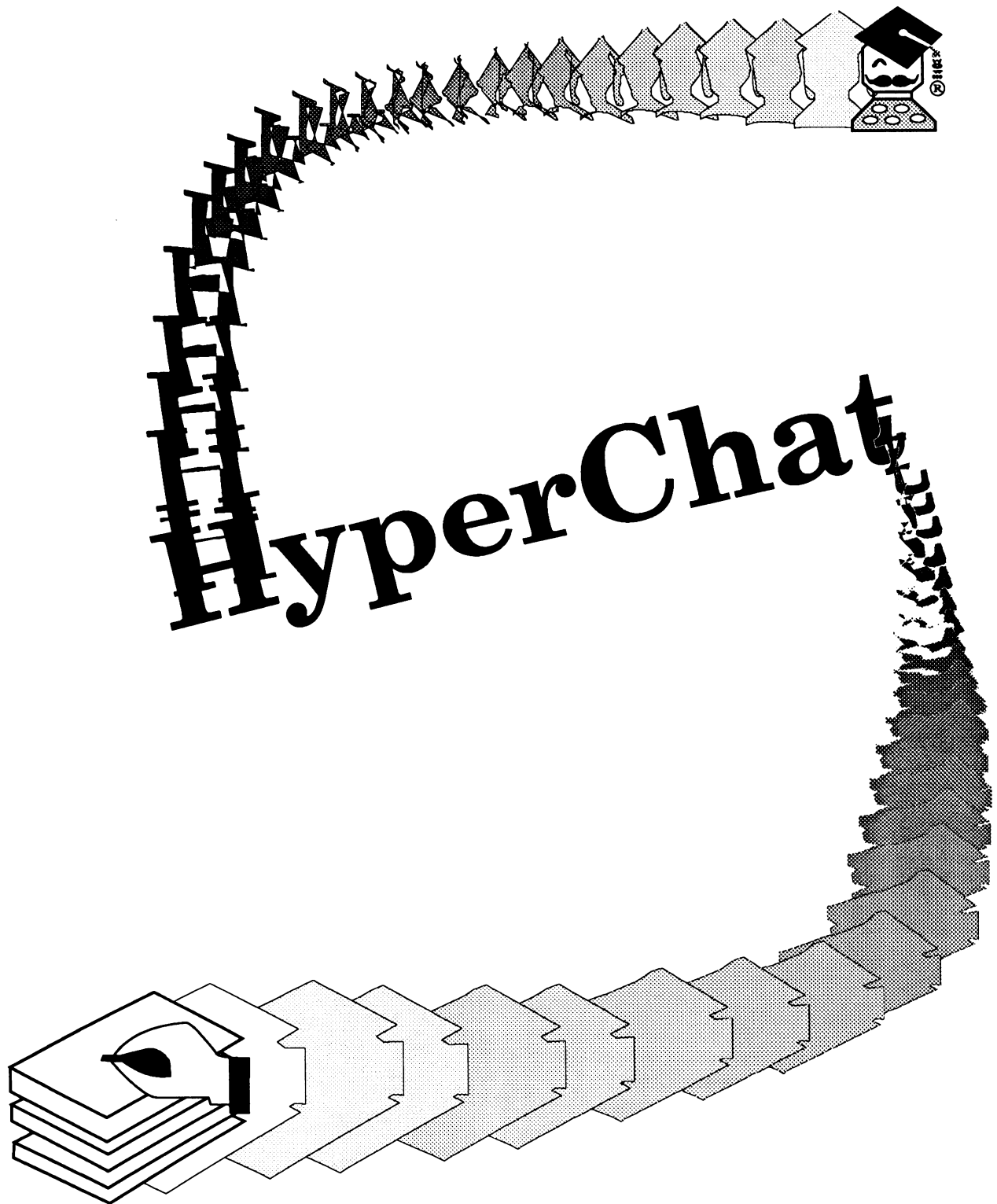
```

```

'*****
'
' o WINDOW ROUTINES o
'*****
'BUILD WINDOW"
IF WND=1 THEN Result=FN GETRESWINDOW(1,1,0)
GOSUB"BUILD EDITS":GOSUB"FILL EDITS":GOSUB "FORMAT WINDOW"
RETURN
'*****
'
' o MENU ROUTINES o
'*****
'BUILD MENU"
Result=FN GETRESMENU(1002,1)
RETURN
'*****
'
' o PUT GRAPHICS & TEXT IN WINDOW o
'*****
'FORMAT WINDOW"
CALL PENNORMAL
:
LONG IF WINDOW(0)=1
PrintString$=FN GETINDSTRING$(31618, 1)
LOCATE 1,1:PRINT PrintString$
PrintString$=FN GETINDSTRING$(31618, 2)
LOCATE 4,2:PRINT PrintString$
END IF
:
CALL PENNORMAL:RETURN
'*****
'
' o CREATE EDIT FIELDS AND BUTTONS o
'*****
'BUILD EDITS"
TEXT ,,0,0
:
LONG IF WINDOW(0)=1
END IF
:
RETURN
'*****
'
' o FILL EDIT FIELDS AND SET BUTTONS o
'*****
'FILL EDITS"
:
LONG IF WINDOW(0)=1
END IF
:
RETURN
'*****
'
' o CAPTURE EDIT FIELD STRINGS & READ BUTTONS o
'*****
'CAPTURE"
:
LONG IF WINDOW(0)=1
END IF
:
RETURN

```





HyperChat™

XCMD Cookbook: HyperAppleTalk



HyperCard

Donald Koscheka
Arthur Young & Company
MacTutor Contributing Editor
Volume 5, Number 1

HyperCard Networking

One of the areas where XCMDs can dramatically alter the capabilities of HyperCard is networking. HyperCard 1.2 introduced the concept of shared stacks by implementing an interface to AppleShare. Unfortunately, too many people have come to believe that AppleShare is all the networking that you need in HyperCard or in any other application on the Macintosh.

Networking can offer a lot more than simple file access to the user. Imagine sending messages or data between stacks interactively. While I was still at Apple, we implemented two stacks that better explain this concept. Imagine that one Mac on the network acted as the stock exchange. Each successive stack that logged in becomes a stock broker and can immediately begin buying and selling stocks and competing against other brokers on the network. This was a fun little project that took on a wonderful new dimension when we added the network interface. There is no truth to the rumor that we began beta testing this stack on October 19, 1987!

A second stack that we wrote provided the capability of an interactive chalkboard. Everybody who logs in can see the chalkboard as well as pick a paint tool and actually draw something on the board. Every other user would see these changes. An interesting exercise in cooperative work area applications.

HyperAppleTalk

The code that I used to implement these sample stacks is a set of XCMDs called "HyperAppleTalk". These XCMDs implemented the AppleTalk Transaction Protocol (ATP) and the Name Binding Protocol (NBP). ATP is a network protocol that implements the server/client model. One entity on the network acts as an information provider or server while subsequent systems act as information requestors or clients. Although this is a very flexible model, the particular implementation of ATP is a little too limiting. Client requests must be less than 578 bytes in length and server responses must be less than 4624 bytes. If a client made a request to the user for a file that is greater than 4600 bytes, the ATP protocol would need help from another protocol layered on top of ATP. I chose not to implement this protocol because ATP as it stands is more than adequate for sharing messages between stacks. You can obtain a copy of these XCMDs, written in MPW C, from the Apple Programmers and Developers Association (APDA).

The second protocol that I implemented in HyperAppleTalk is the Name Binding Protocol. This is the mechanism by which user names are associated with network addresses on AppleTalk. The name binding protocol allows you to register a name with the

network. This is an important feature since network addresses tend to resemble phone numbers. Imagine how much trouble you would have using the phone if you knew only a party's number and not their name!

Over the next several months I will be providing you with network access in HyperCard. This month I will introduce the concept of "connecting" to the network, that is, establishing the address and record structures required to maintain communication with the network. Next month, I will provide a complete set of routines for access to the Name Binding Protocol. The third installment will feature an important new protocol, the AppleTalk Data Stream Protocol. Finally, we'll add some embellishments such as access to multiple networks across bridges and file transfer across the network. The result will be a complete set of protocols that will allow you to interactively send messages, data and even entire files between stacks running on different systems.

"Connecting" to the Network

To help us get started, we need to examine network basics. Because network programming carries the aura of being difficult, I will keep this introduction simple. You can obtain more information from *Inside Macintosh*, *Inside AppleTalk* or from back issues of *MacTutor* (April '87, Vol. 3 No. 4) which is where I first learned how to program for the network.

A network consists of a number of entities that are connected by some common transport mechanism. Foremost of us, this transport mechanism is LocalTalk. A node is any device that can be connected to the network and share information with other devices. For sake of illustration, think of each device on the network as a single node.

The set of software protocols that control the traffic and communications on the network is called "AppleTalk". These protocols are quite independent of the cabling between entities which is why Apple went to such pains to divorce LocalTalk from AppleTalk. This separation pays dividends to the programmer. We can think of AppleTalk as a software mechanism and not worry a whit about what wires are connected.

AppleTalk functions at a variety of levels. At the lowest level, AppleTalk coordinates the transmission and reception of discrete packets of information between entities. To ensure that two entities don't "hog up the wire" each communication is sent as packets of about 600 bytes of information. A 6K file will require about 10 packets to be sent in its entirety. We call the complete communication of all packets a *transaction*. The

completion of a transaction is guaranteed for you by AppleTalk via the Dynamic DataLink Protocol (DDP), and we need not concern ourselves with it at this time (there are lower-level protocols than DDP, but we don't need to know them to do serious programming on the network. Think of the lower layers as the network's "assembly language").

Nodes address each other on the network by their network addresses. As we said, Network Addresses are similar to phone numbers. If you know the network address of any entity, you can communicate with that entity. The network address consists of three objects: the network number, the node and the socket. For now, we'll assume that all entities are on the same network so that the network number is always 0 (we'll relax this constraint later when we introduce the concept of internet addressing with the Zone Information Protocol). The second part of the address is the node id which uniquely identifies an entity within a given network.

The most important concept in the network address is the concept of a socket. A network address is a 16-bit object (a pascal INTEGER) that remains constant for as long as that network remains active. The node id is an 8-byte entity that remains valid for as long as an given node in the network stays up. Sockets, unlike network and node id's, come and go as needed. Sockets are associated with individual transactions on the network and provide a vital way for the network to properly route transactions between nodes on the network.

Socket Flavors

For our purposes, sockets come in two flavors, listening sockets and transaction sockets. Each node on the network needs at least one "semi-permanent" listening socket whose sole job is to react to incoming requests from other nodes. Listening sockets are like the "ringer" on your telephone. They don't do much more than let you know that someone is trying to get a hold of you. Listening sockets are said to be semi-permanent because they are assigned to a single node for as long as that node remains connected to the appropriate protocol. In ATP, a listening socket is equivalent to a server. In ADSP, as we shall see, a listening socket is also called a connection listener.

The second class of sockets are "dynamic" sockets which I refer to as transaction sockets. Once a connection is established between any two nodes, a transaction socket is assigned to those two nodes for the duration of that transaction. Once the transaction is completed, the dynamic socket is returned to the "socket" pool. This is important because a socket is an 8-bit entity. With only 256 possible combinations, the need to conserve and reuse socket numbers becomes important.

The neat thing about dynamic sockets is that they uniquely identify a given transaction. One node can easily carry on multiple conversations simultaneously because each conversation is identified by its unique socket id. Dynamic sockets are assigned and de-assigned automatically. We won't have to concern ourselves with the intricate underworkings of this mechanism in this discussion. All we need to remember is that when we establish a connection with another entity on the network, that connection will receive a unique socket id. When

we "tear down" the connection, that socket will be made available for a future connection.

Name Calling

Keeping track of nodes on the network by their network address is not particularly user friendly; it would be roughly similar to calling people by their social security numbers. And we know that's not user friendly because only government agencies and the DMV would think of doing that!

We need a way of identifying a given node on a given network with a particular user. In other words, we need a way of binding a user-selectable name with a given network address. This is accomplished using AppleTalk's name binding protocol (NBP). To bind a name to a network address, we need both a name and an address. Choosing a name is a matter of taste. You can use the name that the user entered in the chooser, or you can ask the user for a name. We'll present both techniques since not every user types a name in the chooser dialog. Next we'll need to know what address to bind that name to. We can't bind a name to a dynamically assigned address for an obviously subtle reason. That is, a node can have a name, a transaction takes on the name of whichever node the transaction is taking place on. As far as why this is true, it should become obvious as you read through the code (i.e. it is left as an exercise for the student).

The binding of a name to a network address can only be done once we establish an address for the listening socket. The listening socket is protocol-sensitive - we don't want to open a listening socket for ATP if we're going to be using ADSP as our communications protocol. Thus, we'll need to open up access to ADSP first to establish the socket address. Once we have this address, we can pass it to the Name Binding Protocol (NBP) along with a name to establish a link. We refer to the association of a name with a network address as an "entity". An entity is said to be network visible if any other entity on the network can see the address of that entity.

To gain access to the network, we first check to see if AppleTalk is installed with the call to MPPOpen in ADSPOpen (see listing 4). Next, we open up the ADSP protocol driver, initialize some global variables and then establish an ADSP listening socket. We use the setglobal callback to set the values of the globals "HyperADSPData" and "mySocketListener". These globals must be declared before they are used or you may find that earlier versions of HyperCard will bomb out (alas, a bug in HyperCard).

We don't need to open the Name Binding Protocol (NBP) driver since it is automatically opened for us if AppleTalk is running. The XFCN NBPOpen (see listing 5) initializes the fields in the NBPBlock record. Don't worry about what those fields do, they'll be the topic of next month's article.

Listing 1a is a sample HyperTalk script for opening both ADSP and NBP. Note that if ADSP doesn't open, there is no need to open NBP; we didn't get access to the network (the most likely reason for this is that the ADSP driver was not installed at boot time).

If ADSP opens ok, it sets the globaladspdata and mysocket-listener global containers. NBP will need access to the latter

container, while globaladspdata is used solely by our ADSP xcmts. Don't worry about the individual fields in all the records yet, we'll cover them in sufficient detail in upcoming issues.

The important actions to point out right now is the opening of the ".dsp" (aka ADSP) driver and the assigning of a listening socket. Also, three globals will be used for storage. GlobalNBPData stores a pointer to our NBPBlock. Note that the pointer is stored as a numeric string. This is just to keep compatible with HyperCard's wanting to use containers to store strings. Similarly, GlobalADSPData stores a pointer to the ADSPBlock and mySocketListener stores the value of the connection listening socket (to be discussed in gory detail in an upcoming issue). Changing any of these values will result in the untimely death of your stack. Make sure your users don't have access to these containers.

Listings 2-3 contain some global constant and type declarations for ADSP and NBP respectively. Again, there is a lot more there than I can cover in one issue of this magazine, let alone a single column. We will cover all of this information in more detail so stay tuned.

Because we should never leave something undone in HyperCard, a pair of complementary XFCNs are provided in listings 6 and 7. ADSPClose and NBPClose check to see that their global data is allocated. If not, they assume that the drivers are already closed and just return. Otherwise, shut down the communications and clear out the global data. We don't actually close either AppleTalk or ADSP because some other application may have opened them after we did. This is an important exception to the "If you opened, you close it rule". In general, if you open an AppleTalk driver, you can leave it open because another application may already be using it. Listing 1b shows one way to access these XFCNs.

```
on openstack
    global globalNBPData, myRegisteredType, myRegisteredName
    global globaladspData, mySocketListener

    put adspOpen() into it
    if it is empty then
        - have access to adsp and a valid socket in
        - mySocketListener, okay to continue
        put nbpOpen() into it
        if it is not empty then
            answer "Could Not gain access to NBP" with "OK"
        else
            answer "Welcome to AppleTalk" with "OK"
        end if
    else
        answer "ADSP Driver Not installed" with "OK"
    end if
end openstack
```

Listing 1a. A sample Script to Open access to the network

```
on closeStack
    global globalNBPData, myRegisteredType, myRegisteredName
    global globaladspData, mySocketListener

    put adspClose() into some_sort_of_error
    put nbpClose() into yet_another_result
end closeStack
```

Listing 1b. A sample Script for closing down the network

```
(*****)
(* file: ADSPxcmd.p *)
(* *)
(* Interface stuff for adsp *)
(* *)
(* ----- *)
(* By: Donald Koscheka *)
(* Date: 10-Oct-88 *)
(* ----- *)
(*****)

UNIT adspxcmd;

INTERFACE

USES Mtypes, QuickDraw, OSIntf,
    ToolIntf, AppleTalk, ADSP;

CONST

ASYNC = TRUE;
SYNC = FALSE;

(** connection mode status **)
NOP = 0;
REQ = 1;
ACK = 2;
EST = 3;

NBPLSIZE = 120;
ATPBSIZE = 578; (standard size of transaction)

INTERVAL = 20; ( default retry interval)
RETRY = 3; ( retry count =3 (3*60=30 secs))

CLOSE_OK = 0;
RECEIVING = 1;
CLOSE_NOW = 2;

(*****)
(* The following data blocks *)
(* reference memory pointers *)
(* rather than handles. This *)
(* is to insure that all of the *)
(* data in the connection block *)
(* is non-relocatable since *)
(* ADSP runs asynchronously *)
(*****)

TYPE

LIntPtr = ^LongInt;
IntPtr = ^INTEGER;

(***** A connection block *****)
CBPtr = ^Connection;
Connection = Packed Record
next: CBPtr; ( pointer to next connection )
last: CBPtr; ( pointer to last connection )
ccbRef: Integer; ( connection reference # )
mode: Integer; ( set if connection is open )
adr : AddrBlock; ( address of remote end )
msg : Handle; ( callback for incoming dat )
sendQ: Ptr; ( send buffer to remote end )
recvQ: Ptr; ( input buf from remote end )
attn: Ptr; ( attention messages buffer )
outBuf: Handle; ( where outgoing data goes )
inBuf: Handle; ( where the input data goes )
attnBuf: Handle; ( where attention data goes )
remName: Handle; ( entity name of remote end )
```

```

ccb : TRCCB; ( pointer to ccb )
attnPB: DSPParamBlock; ( attention message pb )
dspPB: DSPParamBlock; ( connection pb )
user: Handle; ( place for user data )
End;

```

```

( This is the listening connection )
ADSPPtr = ^ADSPBlock;
ADSPBlock = Packed Record( ADSP protocol data)
  dspRefNum : Integer; ( driver refnum )
  ccbref : Integer; ( listener ccbRefNum )
  addr : AddrBlock; ( listener socket )
  ends : CBPtr; ( connections list )
  ccb : TRCCB; ( listener ccb )
  pb : DSPParamBlock; ( pb for listener )
  chkPoint : Integer; ( set if in callback )
  oldSelf : Byte; ( old self-send flag )
  pad : Byte; ( keep em even )
End;

```

```
Str31 = String[31];
```

```
End.
```

Listing 2. Constant and Type declarations for ADSP connections

```

(*****
(*file: NBPXCMD.p *)
(* *)
(* Constant and type *)
(* declarations for nbp*)
(* xcmts *)
(* *)
(* © Donald Koscheka *)
(*6-October, 1988 *)
(*All Rights Reserved *)
(* *)
(* *)
(* Date | Description*)
(* *)
(* 06-Oct-88| file created *)
(* *)
*****)

UNIT NBPXCMD;

(*****
INTERFACE
*****)

USES MemTypes, QuickDraw, OSIntf, ToolIntf, AppleTalk;

CONST

ASYNC = TRUE;
SYNC = FALSE;
NODE_NAME = -16096; (* name in chooser (STR)*)
MAXNODES = 100; (* maximum nodes for zone*)
NBPLSIZE = 120; (* size local buf for NBP*)
NN = 30; (* #names in lookup table*)
ENTITYSIZE = 110; (* size of entity *)
CLOSE_OK = 0; (* not "CLOSE" critical *)
CLOSE_NOW = 2; (* close down when done *)

TYPE

Str31 = String[31];

NBPB1kPtr = ^NBPBlock;
NBPBlock = RECORD
  Registered: BOOLEAN; (* true = registered*)
  EntCount: INTEGER; (* # of entities visible *)
  LookUpBuffer : Handle; (* lookup buffer *)

```

```

NTEntry : NamesTableEntry; (* entry in names table*)
NBPLocal : array[1..NBPLSIZE] of Char; (* used by NBP*)
END;

```

```
END.
```

Listing 3. Constant and Type declarations for NBP access

```

(*****
(* file: ADSPOpen.p *)
(* *)
(* Open the ".DSP" driver *)
(* and establish memory for *)
(* the ADSPBlock. Creates the *)
(*connection listener *)
(* *)
(* Requires these Globals *)
(* GLOBALADSPDATA *)
(* mySocketListener *)
(* *)
(* DEFINE THE GLOBALS BEFORE *)
(* USING THEM *)
(* *)
(* By: Donald Koscheka *)
(* Date: 10-Oct-88 *)
*****)
Build Sequence

pascal "(adsp)"ADSPOpen.p -o "(hato)"ADSPOpen.p.o
link -m ENTRYPOINT -rt XFCN=1300 -sn Main=ADSPOpen.o
"(hato)"ADSPOpen.p.o
"(libraries)"Interface.o
-o "(hat)"SwitchBoard

(*****
($R-)
($S ADSPOpen)
UNIT Koscheka;

(*****
INTERFACE
*****)

Uses MemTypes, QuickDraw, OSIntf, ToolIntf, PackIntf,
HyperXCmd, AppleTalk, ADSP, adspxcmd;

Procedure EntryPoint( paramPtr : XCmdPtr );

(*****
IMPLEMENTATION
*****)

PROCEDURE ADSPOpen( paramPtr: XCmdPtr ); FORWARD;

Procedure EntryPoint( paramPtr : XCmdPtr );
Begin
  ADSPOpen( paramPtr );
End;

Procedure ADSPOpen( paramPtr : XCmdPtr );
VAR
  refnum : integer;
  tempH : Handle;
  adsp : ADSPPtr;
  error : OSerr;
  mppPB : MPPPParamBlock;
  longStr : Str255;

($I XCmdGlue.inc )

```

```

BEGIN
    error := noErr;
    adsp := NIL; (** assume not in yet**)

    (** Retrieve pointer to our globals **)
    tempH := GetGlobal( 'GLOBALADSPDATA' );

    (** If it in, convert it to a pointer**)
    IF (tempH <> NIL) AND (tempH^ <> NIL) THEN
    BEGIN
        HLock( tempH );
        ZeroToPas( tempH^, longStr );
        adsp := ADSPPtr( StrToLong( longStr ) );
        HUnlock( tempH );
    END;

    IF adsp = NIL THEN
    BEGIN
        error := MPPOpen;

        IF error = noErr THEN
        BEGIN
            (** Open the ADSP Driver **)
            (** ADSP driver must be in **)
            (** your system folder **)
            mppPB.newSelfFlag := 1;
            error := PSetSelfSend( @mppPB, SYNC );

            error := OpenDriver( '.DSP', refnum );
        END;

        (** set up connection listener**)
        IF error = noErr THEN
        BEGIN

            adsp := ADSPPtr( NewPtr( sizeof( ADSPBLOCK ) ) );

            IF adsp <> NIL THEN
            WITH adsp^ DO
            BEGIN
                addr.aNode := 0;
                addr.aNet := 0;
                addr.aSocket := 0;
                ends := NIL;
                dspRefNum := refnum;
                oldSelf := mppPB.oldSelfFlag;
                ccbRef := 0;
                chkPoint := CLOSE_OK;

                WITH pb DO
                BEGIN
                    ( OKAY to open a listening socket )
                    ioCRefNum := refnum;
                    csCode := dspCLInit;
                    ioCompletion := NIL;
                    ccbPtr := @adsp^ccb;
                    localSocket := 0;
                    error := PBControl( @adsp^pb, SYNC );

                    IF error = noErr THEN
                    BEGIN
                        ( Listener is now all ears )
                        ccbRef := ccbRefNum;
                        addr.aSocket := localSocket;
                        csCode := dspCLListen;
                        filterAddress.aNet := 0;
                        filterAddress.aNode := 0;
                        filterAddress.aSocket := 0;
                        error := PBControl( @adsp^pb, ASYNC );
                    END;
                END;
            END; (** with adsp **)

            (** If adsp opened ok, we have a socket**)

```

```

(** so put away the data and the socket**)
IF error = noErr THEN
BEGIN
    SetGlobal( 'GLOBALADSPDATA',
        PasToZero( NumToStr( LongInt( adsp ) ) ) );

    SetGlobal( 'mySocketListener',
        PasToZero( NumToStr( LongInt( adsp^addr ) ) ) );
END
ELSE
    DisposPtr( Ptr( adsp ) );

END
END;

IF error = noErr THEN
    paramPtr^.returnValue := PasToZero( '' )
ELSE
    paramPtr^.returnValue :=
        PasToZero( NumToStr( LongInt( error ) ) );
END;

end.

```

Listing 4. ADSPOpen() XFCN

```

(*****
(*file: NBPOpen.p *)
(* *)
(* Create the control block and*)
(* initialize the data for *)
(* the name binding protocol *)
(* (nbp) stuff. *)
(* *)
(* _____ *)
(* © Donald Koscheka *)
(* 6-October, 1988 *)
(* All Rights Reserved *)
(* *)
(* _____ *)
(* Date | Description *)
(* _____ *)
(* 06-Oct-88 | file created *)
(* _____ *)
*****)

(*****
MPW Build Sequence

pascal "(nbp)"NBPOpen.p -o "(hato)"NBPOpen.p.o
link -m ENTRYPOINT -rt XFCN=2001 -sn Main=NBPOpen
    "(hato)"NBPOpen.p.o
    "(libraries)"Interface.o
    -o "(hat)"SwitchBoard
*****)

($R-)
($S NBPOpen)

UNIT Donald_Koscheka;

(*****
INTERFACE
*****)

Uses MemTypes, QuickDraw, OSIntf,
    ToolIntf, PackIntf, HyperXCmd,
    AppleTalk, nbpxcmd;

Procedure EntryPoint( paramPtr : XCmdPtr );

(*****
IMPLEMENTATION
*****)

```



```
PROCEDURE NBPOpen( paramPtr: XCmdPtr ); FORWARD;
```

```
Procedure EntryPoint( paramPtr : XCmdPtr );
Begin
  NBPOpen( paramPtr );
End;
```

```
Procedure NBPOpen( paramPtr : XCmdPtr );
(*****
* This routine initializes some common
* memory for use by the Name Binding
* Protocol (NBP) Driver and the XCMDs that
* will use it.
*
* YOU MUST CALL THIS ROUTINE BEFORE
* ANY OF THE OTHER XCMDs/XFCNs in the
* AppleTalk suite. Failure to do so
* will result in wildly unpredictable
* and possibly fatal performance by
* hypercard.
*
* IMPORTANT:
* THE HYPERCARD VARIABLE
* "GLOBALNBPDATA" MUST BE
* DECLARED IN YOUR SCRIPT
* BEFORE CALLING NBPOpen
*
* Out: Error Result is returned to hypercard
* or empty if no error
*****)
```

```
VAR
  i : INTEGER; (** just a loop counter**)
  tempH: Handle; (** Used to get global **)
  nbp : NBPBlockPtr; (** MUST NOT MOVE **)
  error: OSerr; (** allocate as pointer**)
  longStr: Str255; (** for ZeroToPas call**)
```

```
($I XCMDGlue.Inc )
```

```
BEGIN
  error := noErr;
  nbp := NIL; (** assume no data yet **)
```

```
(** Retrieve the pointer that's stored **)
tempH := GetGlobal( 'GLOBALNBPDATA' );
```

```
(** If it's defined, convert it to a pointer**)
IF (tempH <> NIL) AND (tempH^ <> NIL) THEN
  BEGIN
```

```
    HLock( tempH );
    ZeroToPas( tempH^, longStr );
    nbp := NBPBlockPtr( StrToLong( longStr ));
    HUnlock( tempH );
  END;
```

```
IF nbp = NIL THEN
  BEGIN
```

```
(** Allocate nbp, initialize its fields**)
  nbp := NBPBlockPtr( NewPtr( sizeof( NBPBlock )));
```

```
IF nbp <> NIL THEN
  WITH nbp^ DO
  BEGIN
```

```
    WITH NTEnter DO
    BEGIN
      nteAddress.aSocket := 0;
      nteAddress.aNode := 0;
      nteAddress.aNet := 0;
```

```
    For i := 1 TO 100 DO
      nteData[i] := chr(0);
```

```
  END;
  (** Clear at the lookup stuff **)
```

```
(** (signifies no lookup yet) **)
  LookUpBuffer := nil;
  EntCount := 0;
  Registered := FALSE;
```

```
  SetGlobal( 'GLOBALNBPDATA', PasToZero(
    NumToStr( LongInt( nbp ) ) ) );
```

```
END
```

```
ELSE
```

```
  error := -1; (** some sort of error occurred **)
END;
```

```
(** Note that no error is reported **)
(** as 'empty', rather than 0. This **)
(** is in keeping with HyperCard **)
IF error = noErr THEN
```

```
  paramPtr^.returnValue := PasToZero( ' ' )
```

```
ELSE
  paramPtr^.returnValue :=
  PasToZero( NumToStr( LongInt( error ) ) );
```

```
END;
```

```
END.
```

Listing 5. NBPOpen() XFCN

```
(*****
(* file: ADSPClose.p *)
(* *)
(* Close down the connection *)
(* listener and all open *)
(* connections. *)
(* Set the globals to empty *)
(* *)
(* Requires these Globals *)
(* GLOBALADSPDATA *)
(* mySocketListener *)
(* *)
(* DEFINE THE GLOBALS BEFORE *)
(* USING THEM *)
(* *)
(* By: Donald Koscheka *)
(* Date: 10-Oct-88 *)
*****)
```

```
(*****
Build Sequence
*****
pascal "(adsp)"ADSPClose.p -o "(hato)"ADSPClose.p.o
link -m ENTRYPOINT -rt XFCN=1301 -sn Main=ADSPClose
"(hato)"ADSPClose.p.o
"(libraries)"Interface.o
-o "(hat)"SwitchBoard
*****
```

```
Build Sequence
```

```
pascal "(adsp)"ADSPClose.p -o "(hato)"ADSPClose.p.o
link -m ENTRYPOINT -rt XFCN=1301 -sn Main=ADSPClose
"(hato)"ADSPClose.p.o
"(libraries)"Interface.o
-o "(hat)"SwitchBoard
*****
```

```
($R-)
($S ADSPClose)
UNIT Koscheka;
```

```
(*****
INTERFACE
*****
```

```
Uses MemTypes, QuickDraw, OSIntf, ToolIntf, PackIntf,
HyperXCmd, AppleTalk, ADSP, adspxcmd;
```

```
Procedure EntryPoint( paramPtr : XCmdPtr );
```

```
(*****
IMPLEMENTATION
*****
```

```
PROCEDURE ADSPClose( paramPtr: XCmdPtr ); FORWARD;
```

```

Procedure EntryPoint( paramPtr : XCmdPtr );
Begin
    ADSPClose( paramPtr );
End;

Procedure ADSPClose( paramPtr : XCmdPtr );
VAR
    tempH : Handle;
    adsp : ADSPPtr;
    error, err : OSErr;
    tpb : DSPParamBlock;
    cb, nb : CBPtr;
    mppPB : mppParamBlock;
    longStr : Str255;

($I XCmdGlue.inc )

BEGIN
    error := noErr;
    adsp := NIL;

    (** Retrieve the pointer **)
    tempH := GetGlobal( 'GLOBALADSPDATA' );

    IF (tempH <> NIL) AND (tempH^ <> NIL) THEN
        BEGIN
            HLock( tempH );
            ZeroToPas( tempH^, longStr );
            adsp := ADSPPtr( StrToLong( longStr ));
            HUnlock( tempH );
        END;

    (** Only kill if allocated **)
    IF adsp <> NIL THEN
        WITH adsp^ DO
            BEGIN
                (* 1 - Tear down connection listener*)
                IF pb.ioResult > 0 THEN
                    WITH tpb DO
                        BEGIN
                            ioCRefNum := dspRefNum;
                            csCode := dspCLRemove;
                            ccbRefnum := adsp^.ccbref;
                            abort := 1;

                            err := PBControl( @tpb, SYNC );
                            IF err <> noErr THEN error := err;
                        END;

                (* 2 - Restore old self send *)
                mppPB.newSelfFlag := adsp^.oldSelf;
                err := PSetSelfSend( @mppPB, SYNC );

                (* PSetSelfSend not implemented on*)
                (* Mac+ (???) , so ignore result*)
                (* "driver can't respond to this *)
                (* control call". *)

                (* 3 - Deallocate all ADSP memory*)
                DisposPtr( Ptr( adsp ) );

                (* 4 * Clear out the containers*)
                SetGlobal( 'GLOBALADSPDATA', PasToZero( '' ));

                (* 5 - leave socket address valid*)
                (*for now *)
            END; (** WITH adsp^ **)

            IF error = noErr THEN
                paramPtr^.returnValue := PasToZero( '' )
            ELSE

```

```

                paramPtr^.returnValue := PasToZero( numToStr( longint( error )) );
            END;
        END.

Listing 6. ADSPClose() XFCN

(*****
(*file: nbpClose.p *)
(* *)
(* Create control block and *)
(* initialize the data for *)
(* the name binding protocol *)
(* (nbp) stuff. *)
(* *)
(* _____ *)
(* © Donald Koscheka *)
(*6-October, 1988 *)
(*All Rights Reserved *)
(* *)
(*****
(*****
    MPW Build Sequence

pascal "(nbp)"nbpClose.p -o "(hato)"nbpClose.p.o
link -m ENTRYPOINT -rt XFCN=2002 -sn Main=nbpClose
    "(hato)"nbpClose.p.od
    "(libraries)"Interface.o d
    -o "(hat)"SwitchBoard
(*****

($R-)
($S nbpClose)
UNIT Donald_Koscheka;

(*****
    INTERFACE
(*****

Uses MemTypes, QuickDraw, OSIntf,
    ToolIntf, PackIntf, HyperXCmd,
    AppleTalk, nbpxcmd;

Procedure EntryPoint( paramPtr : XCmdPtr );

(*****
    IMPLEMENTATION
(*****

PROCEDURE nbpClose( paramPtr: XCmdPtr ); FORWARD;

Procedure EntryPoint( paramPtr : XCmdPtr );
Begin
    nbpClose( paramPtr );
End;

Procedure nbpClose( paramPtr : XCmdPtr );
(*****
    * This routine shuts down access to
    * the NBP driver. It also unregisters
    * the entity if that hasn't already been
    * done.
    *
    * IMPORTANT:
    *
    * THE HYPERCARD VARIABLE
    * "GLOBALNBPDATA" MUST BE
    * DECLARED IN YOUR SCRIPT
    * BEFORE CALLING nbpClose
    *
    * Does not report an error
    *****)
VAR
    error : OSErr;

```

```

longStr : Str255;
tempH : Handle;
nbp : NBPB1kPtr; (** MUST NOT MOVE **)
pb : MPPParamBlock;

($I XCMDGlue.Inc )

BEGIN
    error := noErr;
    nbp := NIL; (** assume no data yet **)

    (** Retrieve the pointer that's stored **)
    tempH := GetGlobal( 'GLOBALNBPDATA' );

    (** If it's there, convert to pointer **)
    IF (tempH <> NIL) AND (tempH^ <> NIL) THEN
        BEGIN
            HLock( tempH );
            ZeroToPas( tempH^, longStr );
            nbp := NBPB1kPtr( StrToLong( longStr ));
            HUnlock( tempH );
        END;

    (** If the data is still allocated, we **)
    (** shut down NBP, otherwise, just quit**)
    IF nbp <> NIL THEN
        WITH nbp^ DO
            BEGIN
                (** Unregister if still registered**)
                IF NTEnter.nteData[1] <> chr(0) THEN
                    BEGIN

```

```

                        pb.entityPtr := @NTEnter.nteData[1];
                        error := PRemoveName( @pb, SYNC );
                    END;

    (** Remove the lookup buffer **)
    IF LookUpBuffer <> NIL THEN
        BEGIN
            HUnlock( LookUpBuffer );
            DisposHandle( LookUpBuffer );
        END;

    (** Dump the NBP Master Block **)
    DisposPtr( Ptr(nbp) );
    SetGlobal( 'GLOBALNBPDATA', PasToZero( '' ));
END;

IF error = noErr THEN
    paramPtr^.returnValue := PasToZero( '' )
ELSE
    paramPtr^.returnValue :=
        PasToZero(numToStr(longint(error)));
END;
END.

```

Listing 7. NBPClose() XFCN

Next month: Accessing the Name Binding Protocol in HyperCard.

end HyperChat



HyperChat™

A Real Window XCMD



Joe Zuffoletto
Cupertino, CA
Volume 5, Number 2

Window XCMD

Part 1: Windows, XCMD's, and MultiFinder

Joe has been at Apple a little over a year, where he works in the Software Quality Assurance department. His group develops automated software test tools. Before coming to Apple, He earned a BS in EE & CS from Princeton University. He has been programming the Mac for about 8 months and spends a lot of time experimenting with HyperCard. This article contains important windowing technology for dealing with gray regions, contributed by Scott Boyd and Greg Marriott.

Of Cards and Windows

For some reason, XCMD's that put up windows intrigue me. I think it's because windows imply event loops, and running an event loop in an XCMD is an interesting idea because it allows you to temporarily "steal the show" from HyperCard. Also, once you throw a window on the screen there's a lot of neat stuff you can put inside it, like graphics too large for HyperCard's card window, files generated by other applications, color pictures, and human interfaces for sophisticated tasks.

In this article I will present an XCMD that displays a document window with scroll bars. This XCMD is full-featured and very robust — essentially a mini-Macintosh application running on top of HyperCard. It is MultiFinder aware, shows how to scroll bitmapped graphics documents, and also demonstrates how to support multiple screens on the Mac II. Finally, it demonstrates how to cope with some human interface issues when it comes to displaying windows on top of HyperCard.

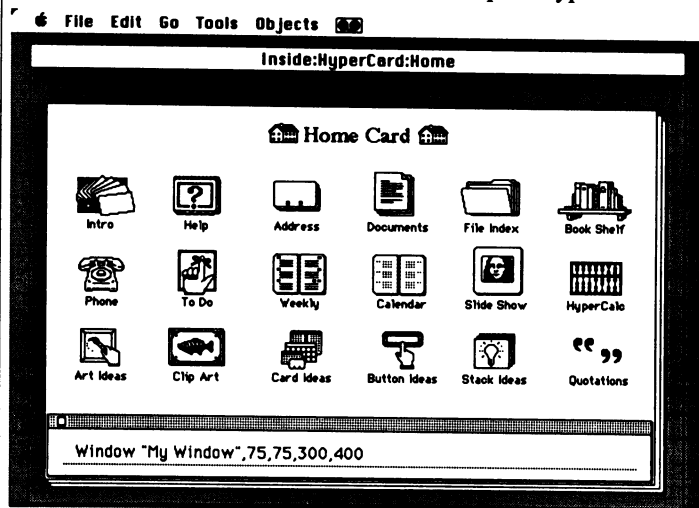


Figure 1a: Before...

My discussion assumes you are somewhat familiar with the issues behind writing XCMD's and XFCN's for HyperCard. In

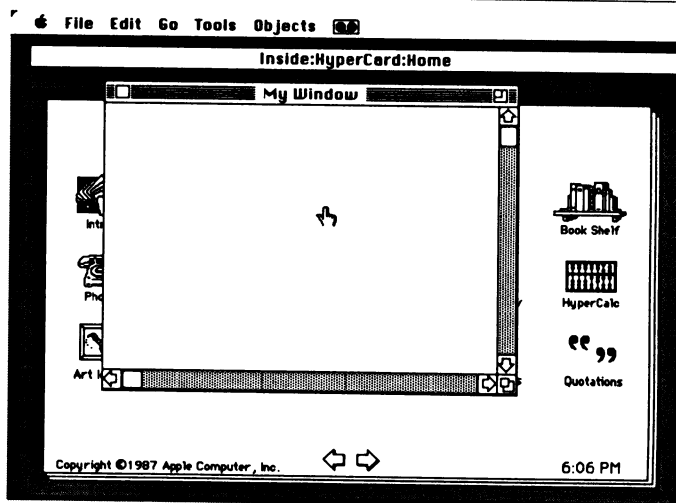


Figure 1b: ...After

particular, I assume you have a nodding acquaintance with the glue routines that perform special functions and facilitate communication between your XCMD and HyperCard. Plenty of introductory articles on the subject have appeared in *MacTutor* over the past few months, and Gary Bond has written an excellent book, *XCMD's for HyperCard* (MIS Press, 1988), which gives you a head start in writing XCMD's (Like Gary, I refer to XCMD's and XFCN's simply as XCMD's).

However, even if you are not writing XCMD's you might learn a lot of useful stuff by reading this article. My XCMD uses an event loop much like one you would find in a normal application, and I show you how to cope effectively with event handling and cursor tracking under MultiFinder. Also, if you'd like to see a rare example of scrolling code, that will be coming next month. In short, there's something here for just about everyone.

One article will not hold all this material, so I'm splitting it into two parts. This month I will show you how to put the window up and deal with MultiFinder, multiple screens, and human interface issues. The window will have scroll bars but they won't work. Next month I'll breathe life into the scroll bars by demonstrating techniques for scrolling bitmapped graphics documents. As a newcomer to Mac programming I feel there is a critical shortage of example scrolling code, so I'd like to add some to the pool.

First Stop: The Drawing Board

I would like to start by describing the design of this fairly extensive XCMD, which I call Window.

The goal is to create a "full-featured" window — one that is draggable, growable, zoomable, and contains scroll bars. Some

other aspects of the design:

- Window is “semi-modal.” This means we cannot use HyperCard while the window is up, but we can switch to other applications under MultiFinder. Boo! Hiss! Yecch! Unfortunately, this “feature” is imposed on us by HyperCard, which, like all Macintosh applications, doesn’t like to share its event loop with strangers.

- Window is “MultiFinder aware;” i.e., the main event loop calls WaitNextEvent and handles suspend/resume events. We don’t need a SIZE resource because we’re running under the auspices of HyperCard and are using whatever space is available in HyperCard’s application heap. Window doesn’t use the clipboard, so it doesn’t need to convert its own private scrap when the user clicks on another open application. This feature could be added, however.

If you’d like more information about programming under MultiFinder, see the “Programmer’s Guide to MultiFinder,” an Apple publication available from APDA.

- Window supports multiple screens on the Mac II. This means the window can be dragged to any screen, grown or zoomed to the size of the largest screen, or grown to span adjacent screens. If the user clicks on the zoom box, the window zooms to whatever screen the zoom box is on at the time, instead of the default behavior of zooming to the main screen.
- Window accounts for human interface anomalies introduced by HyperCard. This means it deals sensibly with HyperCard’s windoids and with the current state of the menubar, which may or may not be visible when Window is invoked. As we’ll see, inattention to the state of the menubar can get you into trouble under MultiFinder!
- The HyperTalk call is:
Window title,top,left,bottom,right
where title is a string (enclosed in double quotes), and the other parameters are integers specifying the window’s screen location in global coordinates; i.e., (0,0) is the top left corner of the screen (or of the main device on a Mac II), with coordinates increasing as you move down and right.
- Window won’t allow you to draw a window whose title bar is off the screen or under the menubar.
- Command-spacebar toggles the menubar off and on, just like in HyperCard. If the menubar is visible and the user clicks the mouse outside the window, the menubar flashes unless the click causes MultiFinder to switch to another open application.
- Clicking on the window’s close box ends the XCMD. Command-W, which closes windows in the Finder and many other applications, is also supported for this purpose.

For now, the window we put up will be empty, but will demonstrate all of the features described above. Next month I’ll add contents to the window when I talk about scrolling.

Plan of Attack

Now that we know the main features of the XCMD, it’s time to think about what steps to take to implement them.

Window, of course, will have an event loop, much like a normal Macintosh application. Because of its semi-modal nature, however, Window will not respond to mouse clicks inside HyperCard’s card window or menubar. Instead, it will just flash the menubar.

Incidentally, we could put up our own menubar and turn Window into a true mini-application, and then restore the menubar to HyperCard’s on exit. As soon as someone thinks of a useful mini-application that runs as an XCMD they will do this, but for now let’s keep things simple and ignore the menubar.

By the way, because we’re ignoring the menubar, what will happen under MultiFinder when the user clicks on the application icon in the upper righthand corner? Will we switch applications? Yes! This is the other side of Window’s semi-modal nature. The only thing that prevents MultiFinder from switching in this situation is if the frontmost window is of the dBoxProc type, which MultiFinder associates with modal dialog boxes. Our window is of type zoomDocProc, so MultiFinder will switch. MultiFinder will also switch if we click on a visible window belonging to a different application.

I want to avoid adding extra resources (besides the XCMD itself) for the user to copy into his or her stack, so I’ll use NewWindow to build my window on the fly instead of pulling it out of a resource with GetNewWindow. This is no big deal because the window is just a standard document window with a zoom box.

As with writing desk accessories, we can have no global variables. This should be no problem.

Early Frustrations

If you have ever tried to put a window on top of HyperCard with an XCMD, you probably experienced all kinds of interesting and unexpected problems. You will therefore recognize some of the obstacles I encountered in early development, which I will now describe to you — solutions included.

The first problem is a complete show-stopper with the following symptoms: The XCMD successfully puts up some kind of window — a dialog box, document window, or whatever — but then bombs consistently on exit. Inspecting the code reveals no obvious problems. What on earth could possibly be wrong? The answer is contained in the following Fundamental Law: *Thou shalt save the current grafPort upon entering thy XCMD and restore it upon leaving.*

When an XCMD starts, the current grafPort is always HyperCard’s card window, and HyperCard expects things to be no different when the XCMD ends. If you set the port to something different, say to a window you put up or to a printing grafPort, and then don’t set it back, HyperCard gets very confused and dances La Bomba.

Let’s now assume the grafPort problem is solved and we have a window up that can be dragged, grown, and zoomed. The next problem pertains to update events; namely, the HyperCard window and the little windoids don’t seem to be getting any.

Instead, when we drag our window on top of them and move it away, big white blotches remain. But under MultiFinder, when we drag our window over other applications and move it away, they update normally. What's going on here?

The answer to this question is fairly interesting, and the described behavior under MultiFinder adds additional insight. First of all, it should be clear why the other applications update normally: It's because MultiFinder sends them update events whenever our window's movements corrupt their displays. Well then, shouldn't MultiFinder be sending HyperCard update events for the same reason? Yes, it should, and indeed it is, but update events for HyperCard are being intercepted by our XCMD's event loop!

At this point we have two choices. We can either throw up our hands and live with the white blotches, or we can figure out a way to pass update events to HyperCard. Following the second path is best, of course, but before you dive into your MultiFinder documentation to figure out how to fake an app4 event, think for a minute about the HyperTalk callback routines. These are routines like ZeroToPas, GetFieldByName, SendHMessage, StrToNum — hey, wait a minute! SendHMessage! That sounds promising.... As you may know, SendHMessage delivers messages from XCMD's to HyperCard (and its cousin, SendCardMessage, delivers messages from XCMD's to the current card). Maybe we can do something like this when we get an update event:

```
VAR
  myWindow: WindowPtr;
  HCPort:   GrafPtr;
BEGIN
  GetPort(HCPort);
  .
  .
CASE myEvent.what OF
  .
  .
  updateEvt:
  BEGIN
    IF WindowPtr(myEvent.message) =
      myWindow THEN
      BEGIN
        SetPort(myWindow);
        BeginUpdate(myWindow);
        DoUpdate(myWindow);
        EndUpdate(myWindow);
      END
    ELSE IF
      WindowPtr(myEvent.message)
      = WindowPtr(HCPort) THEN
      SendHMessage('updateEvt');
    END;
  END;
```

This is the right idea, but SendHMessage takes only legal HyperTalk commands as input parameters! Which HyperTalk command(s) will send an update event to the card window? It turns out that going to any card (including the current card) will do the trick. Since we don't want to switch cards on the user in the middle of our XCMD, let's replace the bogus SendHMessage command above with

```
SendHMessage('go to this card');
```

and, like magic, both our window and the card window will get update events when they need them. But what about the windoids? They still suffer from white-blotchitis.

Egads! Human Interface Issues!

This is where we can be good boys and girls and think about Human Interface Issues. Every Mac programmer should do this at least once in his or her lifetime, and here's a wonderful opportunity to get it over with. We know our window is semi-modal, which implies it is on top of every other window in our layer, including the windoids. We also know that the user can't use any of the windoids while our XCMD is running because we don't know how to process events in them. Despite all this, the windoids remain hilited and appear to be as active and available as ever — a potentially confusing situation for the user. So why don't we hide them?

I happen to think this is a Very Good Idea, and I would like to propose that all HyperCard authors, be they XCMD writers or HyperTalk scripters, observe the following guidelines when creating their products:

- Always hide windoids upon entering event-driven XCMD's or XFCN's.
- Please don't hide windoids from within HyperTalk scripts unless absolutely necessary. Leave the menubar alone, too.
- *Whenever you hide windoids, get and save their current state (i.e., which ones are visible, where they are located on the screen, and which tools/patterns are selected) and return them to that state when you are finished. The same goes for the menubar.*

If I make only one contribution to the Macintosh programming community before I die I hope it is the italicized guideline above. I spend a fair amount of time customizing my HyperCard environment, and I have all the windoids displayed right where I want them. I just can't stand it when someone's script comes along, blows them all away for no apparent reason, and then leaves it to me to put them back. Verry rude. I also can't stand it when people take the menubar away from me and never restore it, especially on a Mac II.

For the purpose of writing event-driven XCMD's, we can satisfy the first and third guidelines with the following code:

```
CONST
  mBarHeight = $BAA; {global}
VAR
  toolVis, menuVis: BOOLEAN;
  toolH: Handle;
  menuBarHeight: INTEGER;
  menuBarHeightPtr: ^INTEGER;
BEGIN
  (Get and save current state of
   windoid)
  toolH := EvalExpr('visible of tool
    window');
  ZeroToPas(toolH, toolStr);
  DisposHandle(toolH);
  toolVis := StrToBool(toolStr);
  IF toolVis THEN
    SendHMessage('hide tool
      window');

  (Get and save current state of
   menuBar)
  menuBarHeightPtr :=
```

```

Pointer(mBarHeight);
menuBarHeight := menuBarHeightPtr;
IF menuBarHeight > 0 THEN
  menuVis := TRUE
ELSE
  menuVis := FALSE;

("Meat" of XCMD goes here")

(Restore windoid when finished)
IF toolVis THEN
  SendHMessage('show tool
                window');
(Restore menubar if you changed it)

```

Of course, we would repeat the tool windoid code for the pattern and message windoids. You don't need to worry about the locations of the windoids because HyperCard "remembers" their positions between hide and show commands.

HyperCard has one other windoid called FatBits that appears whenever you go into fatbits mode, but there are no HyperTalk commands to hide it or show it! Fortunately, Bill Atkinson names all his windoids, and the HyperCard people tell me the names won't change, so all we have to do is walk the window list until we find the window called "FatBits," then hide it if appropriate. I accomplish this in the HideFatBits procedure, which is nested in HideWindoids. Once HideFatBits finds the FatBits window, it saves the WindowPtr to it so we can show it again (if necessary) in the RestoreWindoids routine.

Aack! More Human Interface Issues!!

OK, as long as we're hiding the windoids, why not hide the card window? And since we're ignoring the menubar, we might as well get rid of it too, right?

I don't have strong opinions about either of these ideas. Concerning the card window, I prefer to leave it up for four reasons. First, the current card may contain information we want to see while we are looking at the contents of the window. Second, leaving the card window showing gives us a visual reminder of who owns the window when we are running under MultiFinder, because clicking on either window will bring HyperCard's layer to the front (with our window on top, of course). Third, it gets a little annoying to watch the card window disappear and reappear every time we invoke Window. And fourth, since larger human interface concerns are not at stake here (as they are with the windoids), I'm content to let the user decide in his or her HyperTalk script whether or not the card window will be left showing.

I solve the menubar problem by allowing the user to toggle it with command-space, just like in HyperCard.

Dashing through the Code...

Once I got past the initial obstacles and human interface issues, coding the XCMD was fairly straightforward. In this section I'll zip through the program listing and give you some idea of what's going on. Remember that I'm going to cover the scrolling of bitmaps next month, so all the scrolling and bitmap routines are stubbed out.

Take a look at the Main Program. The first thing we do (after

checking the HyperCard version and saving thy grafPort) is make sure the number of input parameters sent to us is correct by calling CheckParamCount. CheckParamCount tells us nothing about the parameter *types*; if they're mixed up or wrong we could be in trouble.

Assuming the input parameters are cool, we use the standard battery of ZeroToPas and StrToNum calls to convert them to useable form for our program. Then we hide the windoids and draw the window with scroll bars.

Next we enter the main event loop, where we call WaitNextEvent (if available) and pre-process suspend/resume events if MultiFinder feeds them to us. I say "pre-process" because all we do here is set a flag telling us if we are running in the background or not. This information is used later when activate and update events are handled.

A Short Digression: Cursor Tracking with WaitNextEvent

While we're talking about WaitNextEvent, let me say a few words about cursor tracking. Imagine a routine called AdjustCursor which does the following:

1. Get the mouse location in global coordinates;
2. Get the location and size of my window;
3. If the mouse lies within my window's content region (minus the scroll bars), set the cursor to HyperCard's browse tool, and set a region called cursorRgn to region A (see Figure 2a);
4. If the mouse lies outside region A, set the cursor to arrow, and set cursorRgn to region B (see Figure 2b).

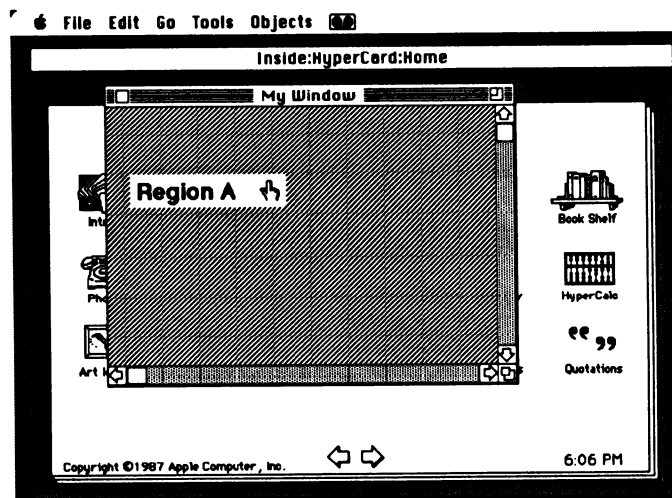


Figure 2a

In the old days (before WaitNextEvent), Macintosh programmers would call a routine similar to AdjustCursor on every pass through the event loop because that was the easiest way to make sure the correct cursor was showing over various parts of a program's human interface. This technique, although reliable, is inefficient because lots of time is spent crunching through AdjustCursor, even when the cursor is not moving.

WaitNextEvent has an interesting feature that makes cursor

tracking more efficient. If I call the hypothetical AdjustCursor routine and then pass a handle to cursorRgn as the last input parameter to WaitNextEvent, I will get a mouse-moved event from MultiFinder when the cursor leaves cursorRgn, and then I can call AdjustCursor again to update the cursor and redefine cursorRgn for next time. This saves me from needlessly updating the cursor on every pass through the event loop.

This is very efficient, but using WaitNextEvent's cursor tracking feature solves only part of the cursor tracking problem. Let me illustrate with an example. Assume I'm using the WaitNextEvent feature, and let's say my window is small and I'm going to zoom it to fill the screen. When I move to the zoom box from region A I'll get a mouse-moved event. Then I'll call AdjustCursor to switch cursorRgn to region B and the cursor to arrow. So far, so good.

Now if I zoom the window and don't move the cursor, the cursor will be sitting somewhere over my window's contents, where it should become the browse tool cursor. But it's still the arrow! Why? Because the old region B is still in effect (instead of a new region A), and a new region A won't be created until I cross into the *old* region A again! To solve this problem, I can call AdjustCursor immediately after zooming the window to account for the window's new size. But I'll experience the same problem after dragging and growing the window, so now I have to call AdjustCursor in these cases as well. Of course, I could just call AdjustCursor when I handle update events, but I still need to call it immediately after dragging the window because dragging doesn't always generate an update event in my layer. I think you can see my point — WaitNextEvent's cursor tracking feature is helpful because it saves you from calling AdjustCursor on every pass through the event loop, but there are a few special cases, mostly related to window movement and sizing, that you must handle yourself. Also, if you are calling GetNextEvent instead of WaitNextEvent, you must still use the old cursor tracking technique (or a more efficient technique of your own invention).

You might have noticed while using HyperCard that the cursor is kind of flaky. In particular, it is sometimes the arrow when it should be the browse tool, but if you move it outside of

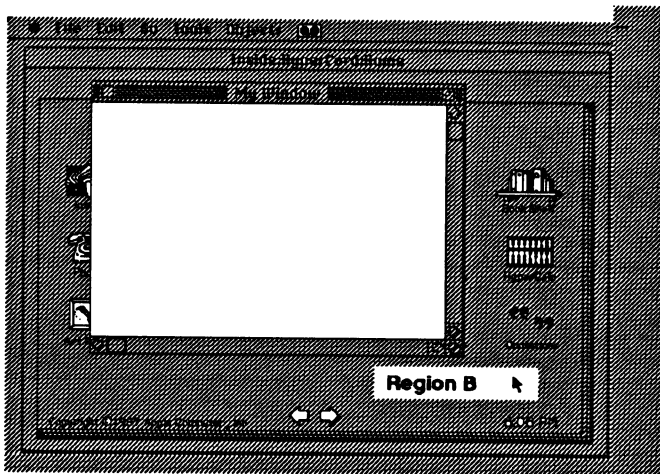


Figure 2B

the card window and then back again, the browse tool reappears! You can probably guess my theory as to why this happens....

The Main Event

OK, we've pre-processed our MultiFinder events and adjusted our cursor. Now we can handle events in our window. Not too many surprises here — I'll zip through each event and summarize what's happening.

MOUSE DOWN:

Flash the menubar (when it is visible) if the mouse click is outside our window; otherwise:

inDrag:

- Drag the window. If you haven't read the Window Manager chapter of *Inside Macintosh*, Volume V, now's a great time to do it. Notice I obtain the dragRect (the rectangle within which we can drag the window) by calling GetGrayRgn, which replaces old-style references to the QuickDraw global screenBits.bounds and automatically takes multiple screens into account for us.
- Update the card window (must do it explicitly because there is no update event in our layer)
- Adjust the cursor region.

inGrow:

- Grow the window. The GrowWindow routine has been modified for multiple screen systems; again, see the Window Manager chapter of *Inside Macintosh*, volume V for details.
- Move the scroll bars

inZoomIn,inZoomOut:

- Zoom the window according to the rules implemented in the ZoomIt procedure.

inContent:

- Deal with scroll bars, if appropriate; otherwise ignore.

inGoAway:

- Exit event loop and XCMD!

ACTIVATE EVENT:

If inBackground is true, then we've just been sent behind another application under MultiFinder. We deactivate our scroll bars and show the menubar if it was hidden. MultiFinder will not take care of the menubar for us, and it's terribly unkind to send the user off to another application without a menubar!

Similarly, when we return, inBackground will be false. We reactivate our scroll bars and return the menubar to whatever state it was in when we left.

UPDATE EVENT:

If inBackground is true, then we have been fed an update event by MultiFinder because the user's mousing around in another application has corrupted our window, HyperCard's card window, or both.

If our window is corrupted, we redraw our deactivated scroll bars and the other stuff in our window.

If HyperCard's window is corrupted, we update Hyper-

Card's window without updating ours. We also call BeginUpdate and EndUpdate to empty the HyperCard window's update region because SendHCMessages alone doesn't do this, and we'll get into an infinite loop of update events if we don't do it ourselves.

If inBackground is false, then we have been fed an update event because our window has either been grown or zoomed.

If our window is corrupted, we redraw our activated scroll bars and the other stuff in the window. Then we adjust the cursor.

If HyperCard's window is corrupted, we update it as described above.

KEYDOWN:

The only other event that concerns us is keyDown. If we get command-space, we toggle the menubar off and on, just like in HyperCard. If we get command-W, we close the window and leave the XCMD.

Sneak Preview

That does it for putting up the window, building an event loop, and dealing with MultiFinder. Next month I'll describe the routines and data structures for scrolling and offscreen bitmaps, which are stubbed out in this month's code listing. I'll show you how to set up an offscreen bitmap, then I'll demonstrate a technique called blitting that produces flicker-free, high-speed scrolling.

Thanks, Everybody!

At this point I would like to mention a few people whose ideas and assistance were of great help to me in this glorious project. At the top of the list is "Mr. Toolbox" himself, Greg Marriott, who responded to most of my questions with an understanding nod and a correct answer. Greg also supplied the code for zooming the window on multiple screens.

Not far behind is "Mr. Window Manager," Scott T Boyd, who taught me how to live without the QuickDraw globals.

[MacTutor readers will be very familiar with both Scott Boyd and Greg Marriott as contributors and friends of MacTutor particularly during their "MacHax days". -Ed]

Thanks to Steve Maller for teaching me how to update HyperCard's window, and to Ginger Jernigan, who took time to review the manuscript.

Thanks to Gary Bond for his excellent book on writing XCMD's, and for the time he took to review the manuscript.

And, gosh, I almost forgot: Thanks, Bill!

(*

© 1988, Apple Computer, Inc.
All rights reserved. Publication rights granted to MacTutor.

Window1.p: A HyperCard XCMD in MPW Pascal 2.0.2
by Joe Zuffoletto
Version 1.0, 29 June 1988

Form: Window title, top, left, bottom, right

Example: window "My Window", 50, 100, 300, 400

Notes: Window puts up a standard document window with scroll bars. The window can be dragged, resized, zoomed, and closed.

Command-W is supported for closing the window. Command-spacebar toggles the menubar on and off, as in HyperCard. If you try to draw the window's title bar off the screen or under the menubar, Window will abort with an error message. Error messages can be examined by looking at HyperCard's global variable "the result" after calling Window.

Window is MultiFinder friendly and works with HyperCard 1.2 or later. It supports multiple displays on the Mac II as well.

You must supply your own code for displaying whatever you want to display in the window.

To compile and link this file using MPW Pascal 2.0.2, select the following lines and press ENTER:

```
Pascal Window1.p
link -o "Hard Disk":HyperCard:"HyperCard Stacks":Home d
      -rt XCMD=2000 -sn Main=Window d
      Window1.p.o (MPW)Libraries:Interface.o d
      (MPW)PLibraries:PasLib.o d
      -m ENTRYPOINT
```

Use other link files as necessary.

The above link directives install the XCMD resource into the Home stack. You can substitute the name of any stack you want; be sure to provide the correct pathname. Also, make sure the target stack already has a resource fork or it won't work. You can create an empty resource fork in a stack with ResEdit.

*)

(\$R-)

(\$S Window)

UNIT DummyUnit;

INTERFACE

USES

MemTypes, QuickDraw, OSIntf, ToolIntf, PasLibIntf, HyperXCmd;

PROCEDURE EntryPoint(paramPtr:XCMDPtr);

IMPLEMENTATION

TYPE Str31 = String[31];

```
OffScrHandle = ^OffScrRecPtr;
(Attach to refCon of a window)
OffScrRecPtr = ^OffScrRecord;
OffScrRecord = RECORD
    (Next month!)
END;
```

```
ScrollHandle = ^ScrollPtr;
(Attach to refCon of a control)
ScrollPtr = ^ScrollRecord;
ScrollRecord = RECORD
    (Next month!)
END;
```

PROCEDURE Window(paramPtr:XCMDPtr); FORWARD;

```

PROCEDURE EntryPoint(paramPtr:XCmdPtr);
BEGIN
  Window(paramPtr);
END;

```

```

FUNCTION Min(int1,int2:INTEGER): INTEGER;
BEGIN
  (Next month!)
END; (Min)

```

```

PROCEDURE InitBlit(theWindow:WindowPtr);
BEGIN
  (Next month!)
END; (InitBlit)

```

```

PROCEDURE InvalContents(theWindow:WindowPtr;
                        theOldSize:Rect);
BEGIN
  (More to come next month!)
  EraseRect(theWindow^.portRect);
  InvalRect(theWindow^.portRect);
END; (InvalContents)

```

```

PROCEDURE DrawContents(theWindow:WindowPtr);
BEGIN
  (Next month!)
END; (DrawContents)

```

```

PROCEDURE ScrollContents(theWindow:WindowPtr;dh,dv:INTEGER);
BEGIN
  (Next month!)
END; (ScrollContents)

```

```

PROCEDURE MyScroll(theControl:ControlHandle;partCode:INTEGER);
BEGIN
  (Next month!)
END; (MyScroll)

```

```

PROCEDURE Window(paramPtr:XCmdPtr);
CONST

```

```

  minParamCount      = 5;
  smallestHeight      = 100;
  smallestWidth       = 100;
  _WaitNextEvent      = $A860;
  _Unimplemented      = $A89F;
  active              = 0;
  inactive             = 255;
  MouseMovedEvt       = $FA;
  SuspendResumeEvt    = $01;
  SuspendEventMask    = $1;
  ConvertScrapMask    = $2;
  browseTool          = 6069;
  HCWidth              = 512;
  HCHeight             = 342;
  padding              = 16;

```

```

VAR
  toolVis,patVis:      BOOLEAN;
  msgVis,fatVis:       BOOLEAN;
  hasWaitNextEvent:    BOOLEAN;
  inBackGround,smallScreen: BOOLEAN;
  DoneFlag,HaveEvent:  BOOLEAN;
  wTop,wLeft,wBottom,wRight: INTEGER;
  partCode,controlCode: INTEGER;
  largestHeight,largestWidth: INTEGER;
  dummy,charCode:      INTEGER;
  screenWidth,screenHeight: INTEGER;
  myDocWidth,myDocHeight: INTEGER;
  eventPoint:          Point;
  wRect,screenRect,dragRect: Rect;
  winSizeLimits:       Rect;
  oldSize:             Rect;
  newSize,dontCare:    LONGINT;
  envError:            OSErr;

```

```

  cursorRgn:           RgnHandle;
  hScroll,vScroll:     ControlHandle;
  whichControl:        ControlHandle;
  myWindow,whichWindow: WindowPtr;
  fatBitsWindow:       WindowPtr;
  HCreFresh:           Str31;
  wT,wL,wB,wR,wTitle:  Str255;
  toolStr,patStr,msgStr: Str255;
  widthStr,heightStr:  Str255;
  myBits:              BitMap;
  theEnv:              SysEnvRec;
  myEvent:             EventRecord;
  wRecord:             WindowRecord;
  HCPort:              GrafPtr;
  theOffScrHandle:     OffScrHandle;
  theScrollHandle:     ScrollHandle;
  myOffScr:            OffScrRecord;
  myScrollRecord:      ScrollRecord;

```

```

FUNCTION TrapAvailable(tNumber: INTEGER; tType:
                        TrapType):BOOLEAN;
  (Check to see if a given trap is implemented.)
BEGIN
  TrapAvailable := NGetTrapAddress(tNumber, tType) <>
    GetTrapAddress(_Unimplemented);
END; (TrapAvailable)

```

```

FUNCTION CreateHScrollBar(theWindow:WindowPtr;
                        theValue,theMin,theMax:INTEGER;
                        theRefCon:LONGINT):ControlHandle;
  (Allocate and draw a horizontal scroll bar in theWindow.
  Return a controlHandle to the scroll bar.)
VAR
  myWindowRect,hScrRect: Rect;
BEGIN
  SetPort(theWindow);
  myWindowRect := theWindow^.portRect;
  SetRect(hScrRect,myWindowRect.left - 1,
    myWindowRect.bottom - 15,
    myWindowRect.right - 14,
    myWindowRect.bottom + 1);
  CreateHScrollBar := NewControl(theWindow,hScrRect,
    'MyHoriz',TRUE,
    theValue,theMin,theMax,
    scrollBarProc,theRefCon);
END; (CreateHScrollBar)

```

```

FUNCTION CreateVScrollBar(theWindow:WindowPtr;
                        theValue,theMin,theMax:INTEGER;
                        theRefCon:LONGINT):ControlHandle;
  (Allocate and draw a vertical scroll bar in theWindow.
  Return a controlHandle to the scroll bar.)
VAR
  myWindowRect,vScrRect: Rect;
BEGIN
  SetPort(theWindow);
  myWindowRect := theWindow^.portRect;
  SetRect(vScrRect,myWindowRect.right - 15,
    myWindowRect.top - 1,
    myWindowRect.right + 1,
    myWindowRect.bottom - 14);
  CreateVScrollBar := NewControl(theWindow,vScrRect,
    'MyVert',TRUE,
    theValue,theMin,theMax,
    scrollBarProc,theRefCon);
END; (CreateVScrollBar)

```

```

PROCEDURE InvalScroll(theWindow:WindowPtr);
  (Accumulate the rectangles occupied by theWindow's
  horizontal and vertical scroll bars into the update
  region.)
VAR
  theRect,tallRect,wideRect: Rect;
BEGIN

```

```

SetPort(theWindow);
theRect := theWindow^.portRect;
ClipRect(theRect);
(Accumulate tallRect, which is occupied by the vertical
scroll bar)
SetRect(tallRect, theRect.right-15,
        theRect.top,
        theRect.right,
        theRect.bottom);
EraseRect(tallRect);
InvalRect(tallRect);
(Accumulate wideRect, which is occupied by the
horizontal scroll bar)
SetRect(wideRect, theRect.left,
        theRect.bottom-15,
        theRect.right,
        theRect.bottom);
EraseRect(wideRect);
InvalRect(wideRect);
END; (InvalScroll)

```

PROCEDURE Deactivate(theWindow:WindowPtr);
 (Deactivate the scroll bars in theWindow in accordance
 with the human interface guidelines. This means we
 must erase everything enclosed by the control rec-
 tangles.)

```

VAR
  theControl:    ControlHandle;
  theControlRect: Rect;
BEGIN
  (I always title my scroll bars 'MyVert' and 'MyHoriz'
  so I can easily find them by walking a window's
  control list.)
  theControl := WindowPeek(theWindow)^(controlList);
  WHILE (theControl <> NIL) DO
    BEGIN
      IF (theControl^.ctrlTitle = 'MyVert') OR
        (theControl^.ctrlTitle = 'MyHoriz') THEN
        BEGIN
          theControlRect := theControl^.ctrlRect;
          InsetRect(theControlRect, 1, 1);
          EraseRect(theControlRect);
          END; (IF)
          theControl := theControl^.nextControl;
        END; (WHILE)
    END; (Deactivate)

```

PROCEDURE HiliteScrollBars(theWindow:WindowPtr);
BEGIN
 (Reactivate the scroll bars; e.g., when we resume
 under MultiFinder. More next month!)

```

  HiliteControl(vScroll, INTEGER(active));
  HiliteControl(hScroll, INTEGER(active));
END; (HiliteScrollBars)

```

PROCEDURE MoveScrollBars(theWindow:WindowPtr);
 (Call this procedure after theWindow has changed size.
 MoveScrollBars erases theWindow's scroll bars, resizes
 them, and redraws them.)

```

VAR
  myWindowRect:    Rect;
  vScrRect, hScrRect: Rect;
BEGIN
  myWindowRect := theWindow^.portRect;
  SetRect(hScrRect, myWindowRect.left - 1,
          myWindowRect.bottom - 15,
          myWindowRect.right - 14,
          myWindowRect.bottom + 1);
  SetRect(vScrRect, myWindowRect.right - 15,
          myWindowRect.top - 1,
          myWindowRect.right + 1,
          myWindowRect.bottom - 14);
  SetPort(theWindow);
  ClipRect(myWindowRect);

```

```

(Hide and resize the scroll bars to fit the new window
size.)
HideControl(hScroll);
HideControl(vScroll);
MoveControl(hScroll, hScrRect.left, hScrRect.top);
SizeControl(hScroll, (hScrRect.right - hScrRect.left),
              (hScrRect.bottom - hScrRect.top));
MoveControl(vScroll, vScrRect.left, vScrRect.top);
SizeControl(vScroll, (vScrRect.right - vScrRect.left),
              (vScrRect.bottom - vScrRect.top));
HiliteScrollBars(theWindow);
ShowControl(hScroll);
ShowControl(vScroll);
END; (MoveScrollBars)

```

PROCEDURE ScrollWithThumb(theControl:ControlHandle;
 theEventPoint:Point);

```

BEGIN
  (Next month!)
END; (ScrollWithThumb)

```

FUNCTION WhichDevice(thePoint:Point):GDHandle;
 (For machines that support color QuickDraw and
 multiple screens, WhichDevice figures out which screen
 thePoint is on and returns a GDHandle to that screen.
 thePoint might be where the mouse was clicked, for
 example. Thanks to Greg Marriott for this code.)

```

VAR
  aDevice:    GDHandle;
  foundOne:    BOOLEAN;
BEGIN
  aDevice := GetDeviceList;
  foundOne := FALSE;
  (Walk the device list until thePoint is contained
  in some device's screen rectangle.)
  WHILE (aDevice <> NIL) AND NOT foundOne DO
    BEGIN
      IF PtInRect(thePoint, aDevice^.gdRect) THEN
        BEGIN
          WhichDevice := aDevice;
          foundOne := TRUE;
          END;
          aDevice := aDevice^.gdNextGD;
        END;
    END; (WhichDevice)

```

FUNCTION MenuBarHeight: INTEGER;
 (Returns the height of the menubar in pixels, as read
 from the low memory global mBarHeight.)

```

CONST
  mBarHeight = $BAA;
VAR
  menuBarHeightPtr: ^INTEGER;
BEGIN
  menuBarHeightPtr := Pointer(mBarHeight);
  MenuBarHeight := menuBarHeightPtr^;
END; (MenuBarHeight)

```

FUNCTION OnAScreen(theRect:Rect):BOOLEAN;
 (OnAScreen returns FALSE if all of a window's title
 bar is off the screen or if any part of it is under
 the menubar. The portRect of the window to be checked
 should be passed in theRect.)

```

CONST
  titleBarHeight = 18;
VAR
  deskRgn:    RgnHandle;
  topLeft, topRight: Point;
BEGIN
  deskRgn := GetGrayRgn;
  topLeft.v := theRect.top - titleBarHeight;
  topLeft.h := theRect.left + titleBarHeight;
  topRight.v := topLeft.v;
  topRight.h := theRect.right - titleBarHeight;

```

```

IF ((PtInRgn(topLeft,deskRgn)) OR
(PtInRgn(topRight,deskRgn))) THEN
  OnAScreen := TRUE
ELSE
  OnAScreen := FALSE;
END; (OnAScreen)

PROCEDURE ZoomIt(theWindow:WindowPtr;partCode:INTEGER;
  clickedWhere:Point);
(ZoomIt supports more elegant window zooming on multiple
screen systems. theWindow will zoom to fill whatever
screen the zoom box was on when it was clicked. The
window state toggles between original size and zoomed
size, as usual. Thanks to Greg Marriott for this code.)
CONST
  titleBarHeight = 18;
TYPE
  WStatePtr = ^WStateData;
  WStateHandle = ^WStatePtr;
VAR
  oldRect,newRect: Rect;
  maxHeight: INTEGER;
BEGIN
  oldRect := theWindow^.portRect;
  IF theEnv.hasColorQD THEN
    BEGIN
      newRect := WhichDevice(clickedWhere)^^.gdRect;
      IF WhichDevice(clickedWhere) = GetMainDevice THEN
        newRect.top := newRect.top + MenuBarHeight;
    END
    newRect := GetGrayRgn^^.rgnBBox;
    newRect.left := newRect.left + 2;
    newRect.top := newRect.top + titleBarHeight + 2;
    newRect.right := newRect.right - 3;
    newRect.bottom := newRect.bottom - 3;
    IF NOT EqualRect(oldRect,newRect) THEN
      WITH WindowPeek(theWindow)^ DO
        WStateHandle(dataHandle)^^.stdState := newRect;
        SetPort(theWindow);
        EraseRect(whichWindow^.portRect);
        InvalRect(whichWindow^.portRect);
        ZoomWindow(theWindow,partCode,FALSE);
      END; (ZoomIt)

($I XCmdGlue.inc)
PROCEDURE Fail(errStr:Str255);
(Fail returns errStr to HyperCard and exits the XCMD.
errStr can then be checked by inspecting HyperCard's
global variable "the result." See "XCMD's for Hyper-
Card" by Gary Bond (MIS Press, 1988) for more details.
© 1988 by Gary Bond
All rights reserved. Publication rights granted MacTutor
You may use this code for NON-COMMERCIAL purposes.)
BEGIN
  paramPtr^.returnValue := PasToZero(errStr);
  SysBeep(1);
  EXIT(Window);
END; (Fail)

PROCEDURE CheckParamCount;
(CheckParamCount sees if the number of parameters
passed to the XCMD matches the number expected. If
not, we exit from the XCMD with an error message.
See "XCMD's for HyperCard" by Gary Bond (MIS Press,
1988) for more details.
© 1988 by Gary Bond
All rights reserved. Publication rights granted MacTutor
You may use this code for NON-COMMERCIAL purposes.)
VAR
  numParams: INTEGER;
BEGIN
  numParams := paramPtr^.paramCount;
  IF(numParams <> minParamCount) THEN

```

```

  Fail('Form: HyperWindow "Window
  Title",top,left,bottom,right');
END; (CheckParamCount)

FUNCTION GetHCVersion: Str255;
(Return a string containing the version of HyperCard
being used; e.g., '1.2')
BEGIN
  ZeroToPas(EvalExpr('the version')^,GetHCVersion);
END; (GetHCVersion)

PROCEDURE HideWindoids;
(Get and save the visible state of the tool, pattern,
message and fatbits windoids; then hide them if they
are showing.)
VAR
  toolH,path,msgH,fatH: Handle;

PROCEDURE HideFatBits;
(HyperCard does not have a built-in command for hiding
and showing the fatbits windoid, so we have to do it
ourselves. HideFatBits walks the window list until it
finds a window with title "FatBits," then hides it if
the visible field of its WindowRecord is true.
HideFatBits also saves the WindowPtr to the fatbits
windoid so we can use it later (e.g., to show the
windoid again.)
CONST
  windowList = $9D6; (Low memory global location.)
VAR
  theWindow: WindowPeek;
  theWindowPtr: ^WindowPtr;
BEGIN
  theWindowPtr := Pointer(windowList);
  theWindow := WindowPeek(theWindowPtr^);
  fatVis := FALSE;
  WHILE (theWindow <> NIL) DO
    BEGIN
      IF (theWindow^.titleHandle^^ = 'FatBits') THEN
        BEGIN
          fatBitsWindow := WindowPtr(theWindow);
          IF (theWindow^.visible = TRUE) THEN
            BEGIN
              fatVis := TRUE;
              HideWindow(fatBitsWindow);
              theWindow := NIL;
            END;
          END;
          IF (theWindow <> NIL) THEN
            theWindow := WindowPeek(theWindow)^^.nextWindow;
          END; (WHILE)
        END; (HideFatBits)

BEGIN (HideWindoids)
  (Get visible state of windoids.)
  toolH := EvalExpr('visible of tool window');
  ZeroToPas(toolH^,toolStr);
  DisposHandle(toolH);
  toolVis := StrToBool(toolStr);
  path := EvalExpr('visible of pattern window');
  ZeroToPas(path^,patStr);
  DisposHandle(path);
  patVis := StrToBool(patStr);
  msgH := EvalExpr('visible of message window');
  ZeroToPas(msgH^,msgStr);
  DisposHandle(msgH);
  msgVis := StrToBool(msgStr);
  (Hide the ones that are showing.)
  HideFatBits;
  IF toolVis THEN
    SendHMessage('hide tool window');
  IF patVis THEN
    SendHMessage('hide pattern window');
  IF msgVis THEN

```

```

    SendHCMessage('hide message window');
END; (HideWindoids)

PROCEDURE ShowWindoids;
(This routine assumes HideWindoids has been called
before. ShowWindoids restores the visible state of
the windoids to that saved by HideWindoids.)
BEGIN
    IF toolVis THEN
        SendHCMessage('show tool window');
    IF patVis THEN
        SendHCMessage('show pattern window');
    IF msgVis THEN
        SendHCMessage('show message window');
    (As in HideWindoids, we must take care of the FatBits
    windoid ourselves.)
    IF fatVis THEN
        BEGIN
            ShowWindow(fatBitsWindow);
            SelectWindow(fatBitsWindow);
        END;
    END; (ShowWindoids)

PROCEDURE ToggleMenuBar;
(Set the visible of the menubar to not the visible of
the menubar.)
BEGIN
    IF MenuBarHeight = 0 THEN
        SendHCMessage('show menubar');
    ELSE
        SendHCMessage('hide menubar');
    END; (ToggleMenuBar)

PROCEDURE GetHCBitMap;
BEGIN
    (Next month!)
END; (GetHCBitMap)

PROCEDURE AdjustCursor;
(AdjustCursor changes cursorRgn to the region that
contains the cursor. As soon as the cursor moves out of
cursorRgn, we get an event and can change the cursor and
cursorRgn again. cursorRgn is either the content region
of our window or the region containing everything BUT the
content region of our window.)
VAR
    mousePt:      Point;
    myWinContRect: Rect;
    myWinContRgn: RgnHandle;
    deskRgn:      RgnHandle;
    handHdl:      CursHandle;
BEGIN
    SetPort(myWindow);
    GetMouse(mousePt);
    LocalToGlobal(mousePt);
    myWinContRgn := NewRgn;
    (Calculate the "work region" of our window, which is its
    content region minus the scroll bars and grow icon.
    This is the region within which we want the cursor to
    change to HyperCard's browse tool, and outside of which
    we want it to be an arrow.)
    WITH WindowPeek(myWindow)^.contRgn^.rgnBBox DO
        SetRect(myWinContRect, left,
                top,
                right - 15,
                bottom - 15);
    RectRgn(myWinContRgn, myWinContRect);
    IF PtInRect(mousePt, myWinContRect) THEN
        BEGIN
            (The cursor is in the work region of our window. Set
            to browse tool.)
            handHdl := GetCursor(browseTool);
            IF (handHdl <> NIL) THEN
                SetCursor(handHdl^^)

```

```

        ELSE
            (Set to arrow if can't find browse tool resource.)
            InitCursor;
            (Set the cursor region equal to our window's work
            region.)
            SetEmptyRgn(cursorRgn);
            CopyRgn(myWinContRgn, cursorRgn);
        END
    ELSE
        BEGIN
            (The cursor is outside our window. Set to arrow.)
            InitCursor;
            (Get the current desktop region.)
            deskRgn := GetGrayRgn;
            (Set cursorRgn to the desktop region's bounding box.
            It is important to add the menu bar area to cursorRgn
            too.)
            SetRectRgn(cursorRgn, deskRgn^^.rgnBBox.left,
                    deskRgn^^.rgnBBox.top,
                    deskRgn^^.rgnBBox.right,
                    deskRgn^^.rgnBBox.bottom);
            (Punch out our window's content region from the big
            region.)
            DiffRgn(cursorRgn, myWinContRgn, cursorRgn);
        END;
        DisposeRgn(myWinContRgn);
    END; (AdjustCursor)

BEGIN (Main Program)
    (Check the HyperCard version. Must be 1.2 or greater.)
    IF GetHCVersion < '1.2' THEN
        Fail('Sorry, must have HyperCard 1.2 or greater.');
```

(Save thy grafPort upon entering!)

```

    GetPort(HCPort);
    (Check and reset our environment.)
    CheckParamCount;
    FlushEvents(everyEvent, 0);
    InitCursor;
    (Find out what kind of machine we're running on.)
    envError := SysEnvirons(1, theEnv);
    IF (envError <> noErr) THEN
        Fail('SysEnvirons call failed.');
```

(Convert HyperTalk input parameters for use here.)

```

    ZeroToPas(paramPtr^.params[1]^, wTitle);
    ZeroToPas(paramPtr^.params[2]^, wT);
    ZeroToPas(paramPtr^.params[3]^, wL);
    ZeroToPas(paramPtr^.params[4]^, wB);
    ZeroToPas(paramPtr^.params[5]^, wR);
    wTop := INTEGER(StrToNum(wT));
    wLeft := INTEGER(StrToNum(wL));
    wBottom := INTEGER(StrToNum(wB));
    wRight := INTEGER(StrToNum(wR));
    (If window size parameters are too small or illegal, set
    the window to a predefined minimum size.)
    IF ((wRight - wLeft) < smallestWidth) THEN
        wRight := wLeft + smallestWidth;
    IF ((wBottom - wTop) < smallestHeight) THEN
        wBottom := wTop + smallestHeight;
    (Make sure the user is not trying to draw the window off
    the screen or under the menubar.)

    SetRect(wRect, wLeft, wTop, wRight, wBottom);
    IF NOT OnAScreen(wRect) THEN
        Fail('You are trying to draw your window off the
        screen!');
```

(Get the bounds of the desktop.)

```

    screenRect := GetGrayRgn^^.rgnBBox;
    (If we have a small screen, make a note of it so we can
    hide the card window during context switches under
    Multifinder.)
    ZeroToPas(EvalExpr('item 3 of the screenRect')^, widthStr);
    screenWidth := INTEGER(StrToNum(widthStr));
    ZeroToPas(EvalExpr('item 4 of the screenRect')^,
    heightStr);

```

```

screenHeight := INTEGER(StrToNum(heightStr));
IF (screenWidth = 512) AND (screenHeight = 342) THEN
  smallScreen := TRUE
ELSE
  smallScreen := FALSE;
HideWindows;
IF MenuBarHeight > 0 THEN
  menuWasHidden := FALSE
ELSE
  menuWasHidden := TRUE;
(This is for scrolling and will be explained next month.)
theOffScrHandle := OffScrHandle(NewHandle
  (SizeOf(OffScrRecord)));
IF MemError <> noErr THEN
  Fail('Out of memory. Buy more.');
```

myDocWidth := HCWidth + padding;
myDocHeight := HCHeight + padding;
WITH theOffScrHandle^^ DO
 BEGIN
 (Next month!)
 END;
 GetHCBtMap;
 (Draw the window!)
 myWindow := NewWindow(@wRecord,wRect,
 wTitle,TRUE,zoomDocProc,
 WindowPtr(-1),TRUE,1);

IF (myWindow = NIL) THEN
 BEGIN
 ShowWindows;
 Fail('Not enough memory to draw window.');

END
ELSE
 BEGIN
 (This is for scrolling. Stay tuned....)
 SetWRefCon(myWindow,LONGINT(theOffScrHandle));
 InitBlit(myWindow);
 DrawGrowIcon(myWindow);
 theScrollHandle := ScrollHandle(NewHandle
 (SizeOf(ScrollRecord)));
 IF MemError <> noErr THEN
 Fail('Out of memory. Buy more.');

(Draw horizontal and vertical scroll bars in our
window.)
hScroll := CreateHScrollBar(myWindow,0,0,
 myDocWidth,
 LONGINT(theScrollHandle));
vScroll := CreateVScrollBar(myWindow,0,0,
 myDocHeight,
 LONGINT(theScrollHandle));
HiliteScrollBars(myWindow);
DrawContents(myWindow);
SetRect(dragRect,screenRect.left + 4,
 screenRect.top,
 screenRect.right - 4,
 screenRect.bottom - 4);
largestHeight := screenRect.bottom - screenRect.top;
largestWidth := screenRect.right - screenRect.left;
SetRect(winSizeLimits,smallestWidth,
 smallestHeight,
 largestWidth,
 largestHeight);
HCRrefresh := 'Go to this card';
cursorRgn := NewRgn;
inBackground := FALSE;
DoneFlag := FALSE;
REPEAT
 (Call WaitNextEvent, if available. Otherwise call
 GetNextEvent.)
 IF hasWaitNextEvent THEN
 HaveEvent := WaitNextEvent(everyEvent,
 myEvent,30,cursorRgn)

ELSE
 BEGIN

```

  HaveEvent := GetNextEvent(everyEvent,myEvent);  

  AdjustCursor;  

END;  

IF HaveEvent THEN  

  BEGIN  

    IF (myEvent.what = app4Evt) THEN  

      (Pre-process app4Evt's fed to us by MultiFinder.)  

      CASE BSR(myEvent.message,24) OF  

        MouseMovedEvt:  

          AdjustCursor;  

        SuspendResumeEvt:  

          BEGIN  

            myEvent.what := activateEvt;  

            (Resume event.)  

            IF (BAND(myEvent.message,SuspendEventMask) <>  

              0) THEN  

              inBackground := FALSE  

            ELSE  

              (Suspend event.)  

              inBackground := TRUE;  

              myEvent.message := LONGINT(myWindow);  

            END; (SuspendResumeEvt)  

          END; (CASE BSR)  

        END; (IF HaveEvent)  

      CASE myEvent.what OF  

        mouseDown:  

          BEGIN  

            partCode := FindWindow(myEvent.where,whichWindow);  

            IF (whichWindow = myWindow) THEN  

              BEGIN  

                (Deal with mouse hits to our window.)  

                CASE partCode OF  

                  inDrag:  

                    BEGIN  

                      SelectWindow(whichWindow); (DragWindow bug)  

                      DragWindow(whichWindow,myEvent.where,  

                        dragRect);  

                      SendCardMessage(HCRrefresh);  

                      AdjustCursor;  

                    END;  

                  inGrow:  

                    IF StillDown THEN (GrowWindow bug)  

                    BEGIN  

                      oldSize := whichWindow^.portRect;  

                      newSize := GrowWindow(whichWindow,  

                        myEvent.where,  

                        winSizeLimits);  

                      IF (newSize <> 0) THEN  

                        BEGIN  

                          InvalScroll(whichWindow);  

                          SizeWindow(whichWindow,  

                            LOWORD(newSize),  

                            HIWORD(newSize),FALSE);  

                          InvalContents(whichWindow,oldSize);  

                          DrawGrowIcon(whichWindow);  

                          MoveScrollBars(whichWindow);  

                        END; (IF newSize)  

                      END; (IF StillDown)  

                    END; (inGrow)  

                  inZoomIn,inZoomOut:  

                    BEGIN  

                      IF (TrackBox(whichWindow,  

                        myEvent.where,  

                        partCode)) THEN  

                        BEGIN  

                          InvalScroll(whichWindow);  

                          ZoomIt(whichWindow,partCode,  

                            myEvent.where);  

                          InvalContents(whichWindow,oldSize);  

                          DrawGrowIcon(whichWindow);  

                          MoveScrollBars(whichWindow);  

                        END; (IF TrackBox)  

                      END; (inZoomIn,inZoomOut)
```

```

inContent:
BEGIN
  SetPort(whichWindow);
  eventPoint := myEvent.where;
  GlobalToLocal(eventPoint);
  controlCode := FindControl(eventPoint,
                             whichWindow,
                             whichControl);
  ClipRect(whichWindow^.portRect);
  IF (controlCode = inThumb) THEN
    ScrollWithThumb(whichControl, eventPoint)
  ELSE IF (controlCode <> 0) THEN
    dummy := TrackControl(whichControl,
                          eventPoint,
                          @MyScroll);

END; (inContent)
inGoAway:
BEGIN
  IF (TrackGoAway(whichWindow, myEvent.where))
  THEN
    DoneFlag := TRUE;
  END; (inGoAway)
END; (CASE partCode)
END; (IF whichWindow = myWindow)
END; (mouseDown)
activateEvt:
BEGIN
  IF (WindowPtr(myEvent.message) = myWindow) THEN
  BEGIN
    SetPort(myWindow);
    ClipRect(myWindow^.portRect);
    DrawGrowIcon(myWindow);
    IF inBackground THEN
    BEGIN
      (We've been sent behind another application
      under MultiFinder, so deactivate the scroll
      bars and show the menubar if it was hidden.)
      Deactivate(myWindow);
      IF smallScreen THEN
        ShowHide(WindowPtr(HCPort), FALSE);
      IF MenuBarHeight = 0 THEN
      BEGIN
        SendHMessage('show menubar');
        menuWasHidden := TRUE;
      END
      ELSE
        menuWasHidden := FALSE;
      END
    ELSE
    BEGIN
      (We've been brought to the front under Multi-
      Finder, so reactivate the scroll bars and
      hide the menubar if it was hidden before.)
      ShowHide(WindowPtr(HCPort), TRUE);
      IF menuWasHidden THEN
        SendHMessage('hide menubar');
        DrawControls(myWindow);
        HiliteScrollbar(myWindow);
        FlushEvents(everyEvent, 0);
      END; (IF inBackground)
    END; (IF WindowPtr)
  END; (activateEvt)
  updateEvt:
  BEGIN
    (Handle updates to our window.)

```

```

  IF (WindowPtr(myEvent.message) = myWindow) THEN
  BEGIN
    (Always do this stuff)
    SetPort(myWindow);
    BeginUpdate(myWindow);
    ClipRect(myWindow^.portRect);
    DrawGrowIcon(myWindow);
    (Always do this stuff under single Finder but
    only if in foreground under MultiFinder)
    IF NOT inBackground THEN
    BEGIN
      HiliteScrollbar(myWindow);
      DrawControls(myWindow);
      AdjustCursor;
    END;
    (Always do this stuff)
    DrawContents(myWindow);
    EndUpdate(myWindow);
  END (IF myEvent.message)
  (Handle updates to HyperCard's card window.)
  ELSE IF (WindowPtr(myEvent.message) =
  WindowPtr(HCPort)) THEN
  BEGIN
    SendHMessage(HCRefresh);
    (Must zero out card window's update region
    ourselves because the SendHMessage call
    doesn't do it, and we'll wind up in an
    infinite loop if we don't.)

    BeginUpdate(WindowPtr(HCPort));
    EndUpdate(WindowPtr(HCPort));
  END; (IF... ELSE)
END; (updateEvt)
keyDown:
BEGIN
  IF (BitAnd(myEvent.modifiers, cmdKey) <> 0) THEN
  BEGIN
    charCode := BitAnd(myEvent.message,
                       charCodeMask);
    (Pressing command-spacebar toggles the menubar.)
    IF (CHR(charCode) = ' ') THEN
      ToggleMenuBar;
    (Pressing command-W closes the window and exits
    the XCMD.)
    IF (CHR(charCode) = 'w') THEN
      DoneFlag := True;
    END;
  END;
END; (CASE myEvent.what)
UNTIL (DoneFlag = True);
(Clean up and get outta here!)
DisposeRgn(cursorRgn);
DisposHandle(Handle(theOffScrHandle));
DisposHandle(Handle(theScrollHandle));
CloseWindow(myWindow);
SendCardMessage(HCRefresh);
ShowWindows;
InitCursor;
FlushEvents(everyEvent, 0);
(Restore thy grafPort)
SetPort(HCPort);
END; (IF myWindow...ELSE)
END; (Main)
END. (Window)

```



HyperChat™

XCMD Corner: HyperAppleTalk, Part II



Don Koscheka
Aurthur Young & Company
MacTutor Contributing Editor
Volume 5, Number 2

Last month I introduced access to AppleTalk from HyperCard. Over the next few months I will continue this theme. Adding a multi-user dimension to your stacks should yield some very interesting and exciting results. This month I will introduce the Name Binding Protocol and leave you with a set of XCMDs that implement the more salient features of NBP.

The Basics

First, a quick review of the basics. Sending information across AppleTalk requires two mechanisms: (1) a way of identifying entities on the network so that any two devices can find each other and (2) a "transaction protocol", some method of sending packets of information between any two entities. AppleTalk implements the first of these mechanism with the Name Binding Protocol (NBP) which serves as a sort of "directory assistance for the network. Several protocols exist for sending information between nodes. One of the earliest is the AppleTalk Transaction Protocol which manages quite well for fixed-size packages. A newer and more sophisticated transaction mechanism has recently been introduced by Apple and is called the AppleTalk Data Stream Protocol (ADSP). This protocol is straightforward and easy to understand, which is why I chose it for this project. If you must have access to the network now, I've implemented the ATP protocol which is available in the "HyperAppleTalk" toolkit from Apple. If you want to get a glimpse of the future of easy-to-program networks, I will be discussing ADSP in greater detail next month.

Naming entities on the network is really a courtesy to the user. AppleTalk does not use names to address entities. Each node on the network is assigned a unique address consisting of a network id, a node id and a socket id. These three elements comprise a "phone number" of sorts. AppleTalk identifies entities on the network by the network address. Like the phone system, when you wish to call someone, you pick up the phone and dial that party's number. With no further intervention (until your party picks up the phone at the other end), the phone system establishes the connection and notifies your party with a ringing telephone.

Because AppleTalk does not care who is at the other end of a connection, identifying a party becomes impossible in all but the smallest of network configurations ($n=2$). Imagine that your local phone directory contained the phone numbers of everyone in your town, but not the names associated with those phone numbers. From the telephone system's point of view, the phone book is complete, every phone number is listed in the phone book. From a human interface perspective, a phone book that listed the numbers but not the names would only be useful to if you didn't care who was at the other end of the line.

The network becomes useful to the end user if some sort of mechanism associates entity names with network addresses. An entity name can be anything you choose. As an example, the chooser dialog allows you to enter a name for your node. Entering a name from the chooser, however, doesn't mean that your name is automatically registered on the network. In fact, AppleTalk ignores this name; it is simply placed in a globally accessible "STR" resource (-16906) and made available to any application that wants to know it. You cannot register the name until you receive an address from the network (Imagine that you are applying for a new phone number from Ma Bell. Once the phone number is assigned, the operator will ask you how you want to be registered in the phone book).

Last Month

Last month's column provided a lot of nebulous code in the form of ADSPOpen and NBPOpen. Presenting this material resulted in a bit of a chicken and egg problem. I wanted to introduce NBP first because you already understand it in another form - directory assistance. In order to use NBP, you need a network address (if you were to look someone up in the phone book and not find a phone number next to their name, you might tend to believe that the phone book is not very useful except as a spelling checker). Thus, I needed to present just enough of the transaction side of the equation to cajole an address out of the network. If you compile last month's code and run it on a system that has ADSP installed, about all you will get for your troubles is the address of a connection listener.

When we invoked adspOpen, we first initialized some records that will be used by our network code. Pointers are indicated because HyperCard is fussy about the state of its heap. Our network code executes asynchronously. This technique allows us to get some semblance of background networking out of the system. In effect, we issue a request to send some information on the network and then go away. When the network can, it will send our information and then notify us when the dialog completes, allowing us to move on without waiting for the communication to complete (which can take a long time on the network).

Memory can't be moving around on us because we have no idea when our information will actually be looked at by the network. Normally, allocating our data in handles and then locking down those handles during a dialog would suffice. But, as the saying goes, "HyperCard abhors a locked handle" and will unceremoniously unlock it if it needs to fix up its heap (once you get the hang of this in XCMD programming you begin to appreciate the beauty of it). This poses no problem since we can force our data not to move by allocating non-relocatable memory

in pointers. This is not inconsistent with Inside Macintosh, and it allows us to leave HyperCard's heap in the state that it expects to find it (HyperCard is quite effective at handling pointers always pushing them to the bottom of the heap).

Registering

After allocating and initializing the data, we ask the network to consider us as a node by telling the network manager that we are capable of listening and responding to connection requests from another entity. This is just like hooking up a telephone in your house. Once the phone is installed and the phone company gives you a phone number, anytime the phone rings, you have a good idea that someone is trying to call you. A connection listener performs the same task, notifying us when another party calls. For the moment, let's leave the phone ringing and get back to the matter of how we identify parties on the line.

If you've ever applied for a telephone number you know the phone company will give you a number and then ask how you want to be listed. NBP provides just this feature on the network. Once you get your network address, you want to tell the network what name you can be reached by. This is why we do the adspopen before the nbpopen. First we apply for a phone number; if we get it we list our name in the phone book using the NBPRegister call to AppleTalk.

The first element in the record tells us whether we are registered or not. If we are, then we probably don't need to register again (would registering under several aliases serve any purpose?) NTEEntry is of type NamesTableEntry, a special AppleTalk type that *glues* names to addresses (figure 1). To register an entity with the network, we must place both its internet address (in this case obtained from last month's ADSPOpen XFCN) as well as the name into the names table entry.

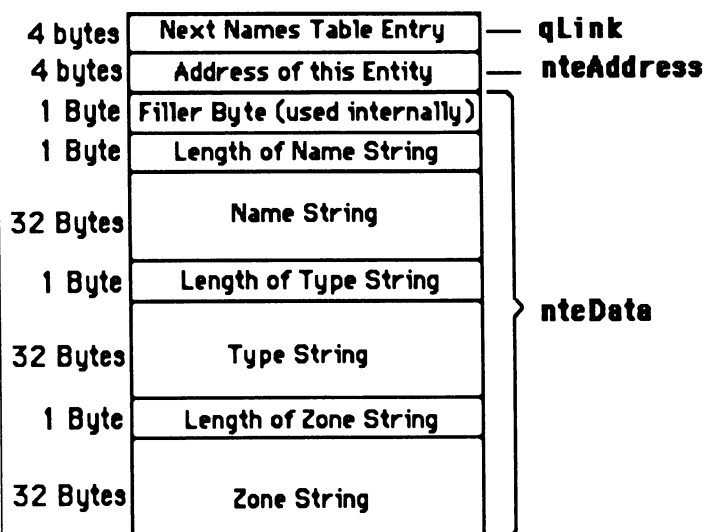


Figure 1. A Names Table Entry

A registered name consists of three parts: (1) zone, (2) name and (3) a type. For now, the zone name is always "*" (we'll relax this constraint when we introduce the concept of inter network addressing in a future article). The name can be anything but

most likely you will use the name already entered in the "User Name" edit box in the chooser.

This is a matter of personal choice and need. Your code might first check the chooser for a name and, if no name is found, then ask the user directly for a name. This HyperTalk script will register the entity using the name in the chooser and setting the type to "HyperPeople":

```
on MouseUp
    global myRegisteredType, myRegisteredName

    Put NBPRegisterName() into errorMessage

    if errorMessage is empty then
        - myRegisteredName now contains the same name
        - as is entered in the chooser
    else
        - report error to user
    end if
end MouseUp
```

The type is quite important. We don't want to send HyperCard messages to entities that are not capable of understanding them so we apply a filter to names: communicate only with entities that are registered as a given type. This makes sense, if you have two entities on the network simultaneously playing chess and checkers you don't want the messages to get crossed (kings, for example, have dramatically different powers in both games). I feel the type is important, so if the user (your stack) doesn't choose one, I set the default to "HyperPeople". This is a type that is capable of reading any HyperCard message whether it makes sense or not!

The rest is pretty easy to figure out. If you know the address of an entity then it should be an easy task to provide that address along with a name and a type to some xcmd that registers that name with the network. NBPRegisterName (Listing 1) does just that. The code is quite straightforward.

The flow of information from node to node in a computer network is governed by the underlying physics of the network. Appletalk's physics may beset be described as a case of organized chaos. I like to think of it as a "mobocratic" system, run by mob rule. While any two entities are passing information between themselves, all other entities must wait for the transaction to complete. The moment the transaction completes, the next entity to get its request on the wire is the winner. It may take several tries to be the next entity on the wire. The network's dynamics govern the number of retries and the time interval between retries; the busier the network, the more you'll have to wait. When you attempt to register with the net, you want to make sure that every entity has an opportunity to update their local copy of the phone book at the time you register. To make sure as many nodes as possible see you register, you specify a retry count. This is just like repeating yourself to a large gathering in hopes that more of the crowd will hear you. Interval specifies the time to wait between retries. The unit time is the "decatick". There are sixty ticks per second so an interval of 6 specifies a waiting period of one second.

The last parameter that we pass to NBPRegisterName is the verify flag. If set true we want the name verified at registration

time. Verification is another convenience item. If any node is already registered with that name, you won't be able to register. The error message is quite explicit so you may wish to ask the user for another name. It's okay to have duplicate names registered, although this can lead to some confusion. Just as when you look up a Smith in the phone book, "Now which Smith is it that I want to call?"

NBPUnRegisterName (listing 2) removes the registered name from the network. This is useful if, for example, you want to register under a new name. The following Script will invoke NBPUnRegisterName:

```
on mouseUp
  global myRegisteredname, myRegisteredType

  Put NbpUnRegisterName() into error
  if error is not empty then put error
end mouseUp
```

Once registered with the network, other entities can see us and send connection requests to us (next month). To be able to see other entities, we need some method of looking up other parties in some sort of directory.

NBPLookupnames (Listing 3) returns a list of the network entities currently visible to us. This is exactly like requesting a phone book from the phone company, except that you will always get the very latest copy. One word of note: the latest copy of the directory may not contain ALL entities on the network, just those that happened to be visible and responded to the update request at the time that the look up was issued.

NBPLookupNames issues a request to find all entities on the network. This function becomes more useful when we limit the process to only entities of a certain type. Since we are really only interested in finding entities that speak HyperTalk, we can filter our request to find only entities of type 'Hyperpeople' (or whatever type you register).

The lookup uses three of the fields in the NBPBlock structure that we introduced last month. EntCount is set to the number of entities that the function actually saw (initially set to zero indicating that no lookup has been performed). We place an upper limit of 100 (maxnodes) entities on the lookup. You can tune this constant to suit the individual needs of your network.

LookupBuffer is a handle to the list of visible entities. You'll use NBPExtractName to pull entities out of the list. NBPLocal is used internally by NBP itself. You provide this buffer to NBP and leave it alone. NBP wants that space and doesn't want us fussing with it. So we don't.

Once the lookup completes, LookupBuffer contains a handle to the entities that were found on the network. The internal structure of the lookup table is somewhat complicated by the intermixing of names and their addresses. To help in extracting information from the table.

This is by no means an exhaustive list of all the capabilities of NBP; but since such a list would truly be exhausting, I've intentionally left some of the NBP routines out of the picture. For the sake of completeness, we will cover these routines as we go ahead with the project. Next month we will introduce the

AppleTalk Data Stream Protocol and provide a stack example that uses these routines.

The data declarations used in these modules are defined in Listing 4. You should refer to last month's article for information on opening a connection listener.

```
(*****
(*file: NBPRegisterName.p *)
(* *)
(* Register a name with the *)
(* network... *)
(* *)
(* Requires that GlobalNBPData be *)
(* initialized and that the global *)
(* myRegisteredName be defined *)
(* *)
(* © 1988, Donald Koscheka *)
(* 10-November, 1988 *)
(* All Rights Reserved *)
(* *)
(*****)

*****
MPW Build Sequence

pascal NBPRegisterName.p -o NBPRegisterName.p.o
link -m ENTRYPOINT -rt XFCN=2003 -sn Main=NBPRegisterName
NBPRegisterName.p.o
  "(libraries)"Interface.o o
  -o YourStackNameHere

*****

($R-)
($S NBPRegisterName)

UNIT Donald_Koscheka;

(*****
INTERFACE
(*****

Uses
  MemTypes, QuickDraw, OSIntf,
  ToolIntf, PackIntf, HyperXCmd,
  AppleTalk, nbpxcmd;

Procedure EntryPoint( paramPtr : XCmdPtr );

(*****
IMPLEMENTATION
(*****

PROCEDURE NBPRegisterName( paramPtr: XCmdPtr ); FORWARD;

Procedure EntryPoint( paramPtr : XCmdPtr );
Begin
  NBPRegisterName( paramPtr );
End;

PROCEDURE NBPRegisterName( paramPtr: XCmdPtr );
(*****
* Register this entity on the network
* using the name and type specified.
*
* Set the global variable, 'myRegisteredName'
* to the name that was registered. Note
* that NBPOpen must be called before
* attempting to register.
*
* params[1]theName
```

```

* Params[2] theType
* params[3]count
* params[4]interval
* params[5]verifyFlag
*
* The type is in the global myRegisteredType
*
* defaults are:
*name = (from chooser)
*type = 'HyperPeople'
*count = 2
*interval = 8
*verify = false;
*
* NOTE: will not register if a socket
* is not open on this node!
*****
VAR
nbp : NBPB1kPtr; (** our global nbp data**)
theName: Str255; (** name to register **)
theType: Str255; (** type for this node **)
theZone: Str255; (** always '(zone name)' **)
str : Str255; (** used in getting globs **)
myName: Handle; (** used for chooser name **)
tempH: Handle; (** used in getting globs **)
count: Byte; (** number of retries**)
interval: Byte; (** wait between retries **)
verify: Byte; (** name must be unique**)
error: OSErr; (** result code **)
i,j : INTEGER; (** pascal string length **)
Mpb : MPPParamBlock; (** param block **)

($I XCMDGlue.Inc )

BEGIN
    error := noErr;
    nbp := NIL;

    (** Retrieve pointer to our NBPData **)
    tempH := GetGlobal( 'GLOBALNBPDATA' );

    (** Convert the string to a handle **)
    IF (tempH <> NIL) THEN
        BEGIN
            HLock( tempH );
            ZeroToPas( tempH^, Str );
            nbp := NBPB1kPtr( StrToLong( Str ) );
            HUnlock( tempH );
            DisposHandle( tempH );
        END;

    IF ( nbp <> NIL ) AND ( NOT nbp^.Registered ) THEN
        BEGIN
            (** Before registering, we need **)
            (** the internet address stored in **)
            (** string form in a container **)
            tempH := GetGlobal( 'mySocketListener' );

            (** Convert the string to a network **)
            (** address (longinteger) **)
            IF ( tempH <> NIL ) THEN
                BEGIN
                    HLock( tempH );
                    ZeroToPas( tempH^, Str );
                    nbp^.NTEEntry.nteAddress := AddrBlock( StrToLong( Str ));
                    HUnlock( tempH );
                    DisposHandle( tempH );
                END;

            nbp^.NTEEntry.qLink := NIL;

            WITH ParamPtr^ DO

```

```

BEGIN
    (** If user specifies a name **)
    (** we'll use it, otherwise use **)
    (** the name in the Chooser **)
    IF ( params[1] = nil ) OR ( params[1]^ = 0 ) THEN
        BEGIN (** Get name from chooser **)
            myName := GetResource( 'STR ', NODE_NAME );

            IF myName <> NIL THEN
                BlockMove( myName^, @theName[0],
                    GetHandleSize( myName ) );
            END
        ELSE
            BEGIN (** Use name passed in **)
                HLock( params[1] );
                ZeroToPas( params[1]^, theName );
                HUnlock( params[1] );
            END;

            (** IF the user provides a type **)
            (** use it otherwise, use the **)
            (** default type **)
            IF ( params[2] = nil ) OR ( params[2]^ = 0 ) THEN
                theType := 'HyperPeople'
            ELSE
                BEGIN
                    HLock( params[2] );
                    ZeroToPas( params[2]^, theType );
                    HUnlock( params[2] );
                END;

            (** AppleTalk, Zone Name must **)
            (** be '*' (this zone) **)
            theZone := '*';

            (** Put the name away in the **)
            (** entity data part of the names **)
            (** table entry. Inside Macintosh **)
            (** vol II p.321 (figure 13) depicts **)
            (** the structure of a names table **)
            (** element. Note that the first **)
            (** byte in the nteData array is used **)
            (** internally. We put the chars **)
            (** away starting at array offset 2 **)

            i := 2;

            FOR j := 0 TO Length( theName ) DO
                BEGIN
                    nbp^.NTEEntry.nteData[i] := theName[j];
                    i := i + 1;
                END;

            (** The type gets tacked to the **)
            (** end of the names string. This **)
            (** is not the same as the concat **)
            (** function, each string keeps **)
            (** its length byte. **)

            FOR j := 0 TO Length( theType ) DO
                BEGIN
                    nbp^.NTEEntry.nteData[i] := TheType[j];
                    i := i + 1;
                END;

            (** Likewise, theZone gets tacked **)
            (** to the end of the type string **)
            FOR j := 0 TO Length( theZone ) DO
                BEGIN
                    nbp^.NTEEntry.nteData[i] := TheZone[j];
                    i := i + 1;
                END;

            (** Number of times to register **)

```

```

IF params[3] <> NIL THEN
BEGIN
  HLock( params[3] );
  ZeroToPas( params[3]^, str );
  count := INTEGER( StrToLong( Str ));
  HUnlock( Params[3] );
END
ELSE
  count := 2;

(** 10X ticks between requests**)
IF params[4] <> NIL THEN
BEGIN
  HLock( params[4] );
  ZeroToPas( params[4]^, str );
  interval := INTEGER( StrToLong( Str ));
  HUnlock( Params[4] );
END
ELSE
  Interval := 8;

(** If the last parameter is true **)
(** make sure that the name is **)
(** not already in use **)
IF params[5] <> NIL THEN
BEGIN
  HLock( params[5] );
  ZeroToPas( params[5]^, str );
  verify := BYTE( strToNum( str ));
  HUnlock( Params[5] );
END
ELSE
  verify := 0;
END;

(** Now set up an make the call **)
(** to register this entity **)
Mpb.ioCompletion := NIL;
Mpb.interval := interval;
Mpb.count := count;
Mpb.entityPtr := @nbp^.NTEnt;
Mpb.verifyFlag := verify;

error := PRegisterName( @Mpb, SYNC);

IF error = noErr THEN
BEGIN
  paramPtr^.returnValue := PasToZero( '' );
  (** As a courtesy to the system,**)
  (** save name off in a global **)
  (** this is an easy way to get **)
  (** name from the chooser **)
  nbp^.Registered := TRUE;
  SetGlobal( 'MyRegisteredName', PasToZero(theName));
END
ELSE
  paramPtr^.returnValue := PasToZero(numToStr(longint(error)));
END;
END;

```

Listing 1. NBPRegisterName

```

(*****)
(*file: NBPUnRegisterName.p *)
(* *)
(* Remove name from the network*)
(* if that name if currently *)
(* registered. *)
(* *)
(* Requires GlobalNBPData be *)

```

```

(* initialized and that global *)
(* myRegisteredName be defined *)
(* *)
(* © 1988, Donald Koscheka *)
(*8-December, 1988 *)
(*All Rights Reserved *)
(* *)
(*****)

(*****
  MPW Build Sequence

pascal NBPUnRegisterName.p -o NBPUnRegisterName.p.o
link -m ENTRYPOINT -rt XFCN=2004 -sn Main=NBPUnRegisterName.o
  NBPUnRegisterName.p.o
  "(libraries)"Interface.o d
  -o yourStackNameHere

*****

($R-)
($S NBPUnRegisterName)

UNIT Donald_Koscheka;

(*****)
  INTERFACE
(*****)

Uses
  MemTypes, QuickDraw, OSIntf,
  ToolIntf, PackIntf, HyperXCmd,
  AppleTalk, nbpxcmd;

Procedure EntryPoint( paramPtr : XCmdPtr );

(*****)
  IMPLEMENTATION
(*****)

PROCEDURE NBPUnRegisterName( paramPtr: XCmdPtr ); FORWARD;

Procedure EntryPoint( paramPtr : XCmdPtr );
Begin
  NBPUnRegisterName( paramPtr );
End;

PROCEDURE NBPUnRegisterName( paramPtr: XCmdPtr );
(*****)
  * UnRegister this entity on the network
  * using the information already contained
  * in the NBP data block.
  *
  * Upon successful completion, clear the
  * global variable, 'myRegisteredName' and
  * set the state of the "Registered" flag
  * for this connection to FALSE.
  *
  *****
  VAR
    nbp : NBPBlockPtr; (** our global nbp data**)
    tempH: Handle; (** used in getting globs **)
    error: OSErr; (** result code **)
    Mpb : MPPParamBlock; (** param block **)
    Str : Str255;

  ($I XCMDGlue.Inc )

  BEGIN
    error := noErr;
    nbp := NIL;

    (** Retrieve pointer to our NBPData **)

```

```

tempH := GetGlobal( 'GLOBALNBPDATA' );

(** Convert the string to a handle **)
IF (tempH <> NIL) THEN
BEGIN
  HLock( tempH );
  ZeroToPas( tempH^, Str );
  nbp := NBPBlkPtr( StrToLong( Str ));
  HUnlock( tempH );
  DisposHandle( tempH );
END;

IF (nbp <> NIL) AND (nbp^.Registered) THEN
BEGIN
  Mpb.ioCompletion := NIL;
  Mpb.entityPtr := @nbp^.NTEntiry.nteData[2];
  error := PRemoveName( @Mpb, SYNC);

  IF error = noErr THEN
  BEGIN
    paramPtr^.returnValue := PasToZero( '' );
    (** As a courtesy to the system, **)
    (** save name off in a global **)
    (** this is an easy way to get **)
    (** name from the chooser **)
    nbp^.Registered := FALSE;
    SetGlobal( 'MyRegisteredName', PasToZero( '' ) );
  END
  ELSE
    paramPtr^.returnValue :=
      PasToZero(numToStr(longint(error)));

  END;
END.

```

Listing 2. NBPUnRegisterName

```

(*****
(*file: NBPLookup.p *)
(* *)
(* params[1] = theType *)
(* params[2] = theZone *)
(* params[3] = number to look up*)
(* params[4] = count *)
(* params[5] = interval *)
(* *)
(* Entity names are returned *)
(* in a list and sent to the *)
(* HyperCard global: *)
(* 'NBPLookUpTable' *)
(* *)
(* ReturnValue is set to the *)
(* result of the lookup *)
(* *)
(* Defaults: *)
(* *)
(* name = '=' ( All names ) *)
(* type = '=' ( all types ) *)
(* zone = '*' (current zone)*)
(* num = 100 ( 100 names ) *)
(* count= 2 ( do 3 lookups ) *)
(* interval= 4( decaticks ) *)
(* *)
(* *)
(* © 1988, Donald Koscheka *)
(*All Rights Reserved *)
(* *)
(*05-Nov-88 *)
(* *)
(*****)

```

```

(*****
MPW Build Sequence

pascal NBPLookupNames.p -o NBPLookupNames.p.o
link -m ENTRYPOINT -rt XFCN=2005 -sn Main=NBPLookupNames.d
NBPLookupNames.p.o
  (libraries)"Interface.o
  (libraries)"PasLib.o
  -o yourStackNameHere

*****

($R-)
($S NBPLookupNames)

UNIT Donald_Koscheka;

(*****
INTERFACE
(*****

Uses
  MemTypes, QuickDraw, OSIntf,
  ToolIntf, PackIntf, HyperXCmd,
  AppleTalk, nbpxcmd;

Procedure EntryPoint( paramPtr : XCmdPtr );

(*****
IMPLEMENTATION
(*****

CONST
  DEFAULT_ERR= 128; (** some sort of mem error **)

TYPE
  Str255Ptr = ^Str255;

PROCEDURE NBPLookupNames( paramPtr: XCmdPtr ); FORWARD;

Procedure EntryPoint( paramPtr : XCmdPtr );
Begin
  NBPLookupNames( paramPtr );
End;

PROCEDURE NBPLookupNames( paramPtr: XCmdPtr );
(*****
* Lookup entities of the
* requested type and zone
* (up to num elements):
*
*****
VAR
  tempH : Handle;
  nbp : NBPBlkPtr;
  EntSize : LongInt;
  i,j,cnt, err,
  intr, total,
  num : INTEGER;
  str : Str255;
  ent : NamesTableEntry;
  Mpb : MPPPParamBlock;
  theType,
  theZone : Str255;

($I XCMDGlue.Inc )

FUNCTION ReturnNames: Handle;
(*****
* Return a list of entities as
* found in the lookup table
*
*****
VAR

```

```

eAddr : AddrBlock; (** Needed by the extract      ***)
newHand: Handle; (** current entity name&typ      ***)
theTable: Handle; (** Lookup Data to return
***)
ret : String[1]; (** newline after each ent      ***)
oldSize: LongInt; (** Previous size of table      ***)
newSize: LongInt; (** size to add to table
***)
i : INTEGER; (** loop control variable    ***)
theEnt: EntityName; (** names from the names table**)

BEGIN
  oldSize := 1;
  theTable := NewHandle( oldSize );

  IF theTable <> NIL THEN
    BEGIN
      theTable^ := 0; (** set theTable to EMPTY**)
      ret[0] := CHR( 1 );
      ret[1] := CHR( 13);

      FOR i := 1 TO nbp^.EntCount DO
        BEGIN
          (** extract name i from the list**)
          WITH nbp^ DO
            BEGIN
              HLock( LookUpBuffer );
              err := NBPEXtract( LookUpBuffer^,
                                total, i, theEnt, eAddr);
              HUnlock( LookUpBuffer );
            END;

            IF err = noErr THEN
              BEGIN
                (** each line in the list gets**)
                (** entities name and type **)
                Str := '';

                Str := Concat( theEnt.objStr,
                              ', ', theEnt.typeStr, ret);
                newHand := PasToZero( Str );
                newSize := GetHandleSize( newHand );
                oldSize := GetHandleSize( theTable );
                SetHandleSize( theTable, newSize + oldSize );
                BlockMove( newHand^,
                           Ptr(ORD(theTable^)+oldSize-1),
                           NewSize);
                DisposHandle( newHand );
              END;
            END; (** FOR i := 1 to Total **)

            (** make sure return string is null **)
            (** terminated or hypercard will have**)
            (** a fit **)
          END; (** IF theTable <> NIL **)
          ReturnNames := theTable;
        END;

      BEGIN
        err := noErr;

        (** Retrieve pointer to our NBPData **)
        tempH := GetGlobal( 'GLOBALNBPDATA' );

        (** Convert the string to a handle **)
        IF (tempH <> NIL) THEN
          BEGIN
            HLock( tempH );
            ZeroToPas( tempH^, Str );
            nbp := NBPB1kPtr( StrToLong( Str ));
            HUnlock( tempH );
            DisposHandle( tempH );
          END;
        END;
      END;
    END;
  END;

```

```

IF ( nbp <> NIL ) THEN (** okay to lookup **)
  BEGIN
    WITH paramPtr^ DO
      BEGIN
        IF params[1] = NIL THEN
          theType := '='
        ELSE
          BEGIN
            HLock( params[1] );
            ZeroToPas( params[1]^, theType );
            HUnlock( params[1] );
          END;

          IF params[2] = NIL THEN
            theZone := '*'
          ELSE
            BEGIN
              HLock( params[2] );
              ZeroToPas( params[2]^, theZone );
              HUnlock( params[2] );
            END;

          IF params[3] <> NIL THEN
            BEGIN
              HLock( params[3] );
              ZeroToPas( params[3]^, Str );
              num := INTEGER(StrToNum( Str ));
              HUnlock( params[3] );
            END
          ELSE
            num := MAXNODES;

          IF params[4] <> NIL THEN
            BEGIN
              HLock( params[4] );
              ZeroToPas( params[4]^, Str );
              cnt := INTEGER(StrToNum( Str ));
              HUnlock( params[4] );
            END
          ELSE
            cnt := 2;

          IF params[5] <> NIL THEN
            BEGIN
              HLock( params[5] );
              ZeroToPas( params[5]^, Str );
              intr := INTEGER( StrToNum( Str ));
              HUnlock( params[5] );
            END
          ELSE
            intr := 4;
          END; (** with paramPtr^ **)

          (** Concatenate the name, type and zone string **)
          ent.ntData[2] := CHR(1);
          ent.ntData[3] := '=';
          i := 4;

          FOR j := 0 TO Length( theType ) DO
            BEGIN
              ent.ntData[i] := TheType[j];
              i := i + 1;
            END;

            (** Likewise, theZone gets tacked**)
            (** to the end of the type string**)
            FOR j := 0 TO Length( theZone ) DO
              BEGIN
                ent.ntData[i] := TheZone[j];
                i := i + 1;
              END;

              (** The lookup data is stored as a handle in **)
              (** our NBPBlock. Since a new lookup supercedes**)
            END;
          END;
        END;
      END;
    END;
  END;

```

```

(** the previous data, delete previous (if any)**)
IF nbp^.LookupBuffer <> NIL THEN
BEGIN
  HUnlock( nbp^.LookupBuffer );
  DisposHandle( nbp^.LookupBuffer );
END;

nbp^.EntCount := 0;
EntSize := sizeof( NamesTableEntry );
nbp^.LookupBuffer:= NewHandle( num * EntSize );

IF nbp^.LookupBuffer <> NIL THEN
BEGIN
  MoveHH( nbp^.LookupBuffer );
  HLock( nbp^.LookupBuffer );

  WITH mpb DO
  BEGIN
    EntityPtr := @ent.nteData[2];
    retBuffPtr := nbp^.LookupBuffer^;
    retBuffSize:= INTEGER( EntSize*num );
    maxToGet := num;
    interval := intr;
    count := cnt;
  END; (** with mpb **)
  err := PLookupName( @Mpb, SYNC );

  IF err = noErr THEN
  BEGIN
    nbp^.EntCount := Mpb.numGotten;
    SetGlobal( 'NBPLookupTable', ReturnNames );
    paramPtr^.returnValue := PasToZero( '' );
  END
  ELSE
    paramPtr^.returnValue :=
      PasToZero(numToStr(longint(err)));

  HUnlock( nbp^.LookupBuffer );
END
ELSE
  err := DEFAULT_ERR; (** no room in the heap? **)
END;
END.

```

Listing 3. NBPLookupNames

```

(*****
(*file: NBPXCMD.p *)
(* *)
(* Constant and type declaration*)
(* file for nbp xcmts *)
(* _____ *)
(* © Donald Koscheka *)
(*6-October, 1988 *)
(*All Rights Reserved *)
(* *)
(* _____ *)
(*****)

UNIT NBPXCMD;

(*****
INTERFACE
(*****)

USES Memtypes, QuickDraw, OSIntf, ToolIntf, AppleTalk;

CONST

  ASYNC = TRUE;
  SYNC = FALSE;
  NODE_NAME = -16096; (* STR resource name from Chooser *)
  MAXNODES = 100; (* maximum # of nodes for zone *)
  NBPLSIZE = 120; (* size of a local buffer for NBP *)
  NN = 30; (* # of names in lookup table*)
  ENTITYSIZE = 110; (* size of entity in lookupbuffer*)

  TYPE

  Str31 = String[31];

  NBPB1kPtr = ^NBPBlock;
  NBPBlock = RECORD
    Registered : BOOLEAN; (* true = registered *)
    EntCount : INTEGER; (* # of entities visible *)
    LookupBuffer: Handle; (* lookup buffer *)
    NTEnt : NamesTableEntry; (* entry in names table *)
    NBPLocal : array[1..NBPLSIZE] of Char;(* used by NBP *)
  END;
END.

```

Listing 4. Definitions needed for the XFCNs



HyperChat™

AppleTalk DSP Part III



Donald Koscheka
Arthur Young & Company
MacTutor Contributing Editor
Volume 5, Number 3

Before I start with this month's column, I would like to explain the change in this month's byline:

Prior to joining Arthur Young in New York, I worked for Apple Computer Inc in Cupertino. When our daughter, Colleen, was born a year ago, my wife and I agreed that we would move back east so that Colleen would have an opportunity to get to know both her paternal and maternal grandparents who live in the NY area. My challenge was to find an interesting programming job, specializing on the Macintosh, in the New York area. My first inclination was to see whether Apple would be interested in putting someone with my experience in the field, perhaps to support large accounts in their Macintosh projects.

Unfortunately, this proved to be a dead end. I was told that Apple needed engineers with IBM experience, Macintosh experience was not needed nor even welcome outside of Cupertino. In my despair, I realized that I was going to have to look for a position outside of Apple.

Interestingly enough, my job search never went farther than the pages of MacTutor. Arthur Young and Company, a large accounting firm based in New York, placed an ad in MacTutor inviting interested parties to interview with the firm's Macintosh development group.

Arthur Young is one of the world's largest users of the Macintosh; their reputation for ingenious uses of the Macintosh and their professionalism spurred me to respond to the ad. As you can tell by the byline, the rest is history.

The lesson: Whether you have a product to sell or you have an interesting programming solution to share with the rest of us, MacTutor is the best forum for reaching the Mac Programming community.

Now this month's column:

If you've been following this column for the last several months, you know that we are building an interface to AppleTalk in HyperCard. In the first installment, I introduced some network basics as well as a few routines to open and close the AppleTalk Data Stream Protocol (ADSP) driver. The second installment featured an interface to the Name Binding Protocol (NBP).

This month, I would like to present the bulk of the routines that will be needed to support ADSP from Hypercard. Due to space considerations, I will present the Hypercard XCMD glue for these routines in next month's issue. I apologize for the seemingly capricious way that I have broken up this information, but I could easily fill a book with this material - programming the network is fun, and I have a lot of information to cover.

Listing 1 contains the file "ADSPXCMP.P" which contains the data definitions for the adsp interface. Listing 2 contains the file "XCMDADSP.INC". This file should be included in your

XCMD immediately following the "XCMDGlue.Inc" declaration. It contains the underlying code used in our appleTalk interface.

ADSP is a very elegant protocol that can best be described in terms of the phone system. In order to call someone, you first look up their number. Next you dial the phone and wait for a connection. Once the connection is established, you begin talking.

I chose the AppleTalk Data Stream Protocol (ADSP) as the transport mechanism because of its intuitive programming model. We will use ADSP to send HyperCard messages between stacks that are running on separate machines. These machines will refer to each other as the "remote end", that is, the other side of the connection.

ADSP is a new protocol that provides "peer to peer" communications on the network. Any two entities that communicate with each other share the same capabilities. Contrast this with the "server-client" model where clients request information and servers provide it. Peer to peer models have the advantage of being bidirectional. Either party may request or provide information. In fact, one can easily model the "client/server" protocol using peer-to-peer communications.

The routines in listing 2 support four parts of an ADSP interface: (1) Looking through the phone book for a number or name (NBPGetAddress and NBPGetName), (2) Hangup on someone (DSPKillData, DSPDisconnect and DSPHangUp), (3) Dial and connect (DSPAddConnection, DSPCreate) and (4) Talk to another party (DSPTalk). In case you've noticed that I have not provided any code for listening, I will present ADSPListen in its entirety next month. Listening on the network, like listening in general, is the more difficult part of communications. One of the reasons for presenting the routines in Listing 2 before presenting the XCMDs that use them is that ADSPListen shares a number of these routines with other XCMDs. A review of the first two issues of this column might help you follow Listing 2 a little better.

Let your fingers do the walking

Transaction protocols such as ADSP expect to work with network addresses. Both NBPGetAddress and NBPGetName provide the name binding support needed by ADSP to determine the address of an entity given that entity's name and vice versa.

NBPGetAddress and NBPGetName let your fingers do the walking. Given a currently visible name, NBPGetAddress will return the "phone number". Conversely, NBPGetName returns the name given the address. The latter is useful when a remote entity is calling in. Both routines expect to have a list of network visible entities available in the lookupBuffer field of the nbp data

block. If this field is empty, then no lookup has been done. No name/address binding can occur until a lookup is performed. A good rule of thumb is to invoke `NBPLookupNames` prior to calling anyone on the network to make sure that the names table is valid. Periodic updates to the names table are also a good idea. A good time to perform the periodic lookup is when you receive an incoming call from an entity that `NBPGetNames` can't find in the lookup table (the lookup table is "stale").

Operator, Get me Mr. Winkler

In order to call another entity, you must establish a connection with that entity. Typically, you call an entity by name. This is equivalent to an old-fashioned telephone exchange where you ring up the operator and said something to the effect, "Hello, operator, this is Dennis Mitchell. Can you connect me to Mr. Wilson?"

`DSPCreate` acts as the operator and will attempt to establish a connection.

First we allocate room for the connection data in the heap via a `newPtr` call. If we get the connection block (cb) we go ahead and initialize its fields. The only space we must allocate *a priori* is room for the input and output buffers. All the other fields get set to NIL or 0.

`DSPAddConnection` puts the connection block into the linked list. This is standard linked list code and allows us to maintain more than one connection at a time.

We interface to ADSP via parameter block I/O. The parameter block is stored in the `dspPB` field of the connection block. Parameter block I/O is the lowest programming level that the application programmer usually interfaces with devices on the Macintosh. Managers such as the file manager and the print manager come with an extra layer of "glue" that shields the programmer from the parameter block. At the time of this writing, no such glue existed for ADSP, you interface with it via the parameter block.

All calls to adsp are made via the `PBControl` call with the `csCode` field set to the operation to perform. In the case of `DSPCreate`, we wish to establish a connection with the remote end. The `csCode` for establishing a connection is `dspInit`. Interfacing to AppleTalk via parameter blocks is Apple's preferred method and, while it takes a little getting used to, it's not really all that hard to follow. One hint: the `csCode` field of the parameter block will tell you what operation is being requested.

Sorry, Bad Connection

`DSPHangUp` undoes the work of `DSPCreate`. It first requests ADSP to remove, or tear down, the connection. This process will notify the remote end that the connection is going down. The `dspRemove` request is performed synchronously as we must wait for it to complete before we can deallocate the data used by the connection.

`DSPDisconnect` removes the connection from our list of connections, sends a message to Hypercard advising of the disconnected line and then calls `DSPKillData` to remove the data associated with this connection. You may not want to send a disconnect notice to Hypercard so either set `remName` to NIL or

simply remove that portion of the code.

Once the connection is removed from the connection list, `DSPKillData` deallocates all of the data associated with a connection. The connection data includes an input buffer, an output buffer and an attention buffer. We do not use the attention buffer in this incarnation.

Don't you ever stop talking?

The last routine of note in listing 2 is `DSPTalk`. This is a simple routine that allows you to send data over an established connection. Note that the connection block is already assumed to be valid, some other routine must have performed the cross-check, as we shall see next month.

Sending information is remarkably simple in ADSP: first copy the data to an output buffer in the connection's data block. Next, issue a `dspWrite` request to the remote end. Here, the transmission is performed asynchronously. We can let the transmission take as long as it needs allowing control of the system to be almost instantaneously returned to Hypercard. Next month I will show you how to detect the completion of a transmission.

Support Routines

In addition to the ADSP routines, `XCMDADSP` includes a number of routines that are needed throughout the interface:

`Retrievedata` accepts the name of a hypercard global container and converts its contents to a long integer. It is used to retrieve the pointers to the NBP and ADSP data blocks created by `NBPOpen` and `ADSPOpen` respectively.

`StringtoShort` converts a zero-terminated string to a pascal integer type. Hypercard parameters are passed as string representation of numbers. Several parameters to the ADSP routines require integer arguments.

`StrCmp` compares two pascal-style strings. It is used by the NBP `getAddress` procedure to match the input string to the strings in the lookup table. When the entity name matches the input string, the ensuing address can easily be extracted from the table (the "phone book").

This is a lot of code to digest in one month but once you start writing stacks that can interact with each other, I think you'll find that staying with me for one more month is well worth the effort. Moreover, I believe that ADSP represents the type of network interface that takes the pain and drudgery out of networking. A lot of the code that I have presented here can be used outside of Hypercard. I hope you give ADSP a try, you might find its full capabilities quite intriguing!

Next month: We wrap up our initial interface to AppleTalk with four XFCNS: `ADSPCall`, `ADSPHangup`, `ADSPTalk`, `ADSPListen`.

Listing 1:

```
(*****)
(*file:   ADSPxcmd.p                      *)
(*version: .01β                          *)
(*                                           *)
```

```

(* ----- *)
(* By: Donald Koscheka and Nancy Rosenberg *)
(* Date: 1-Feb-88 *)
(* © Copyright 1988, Apple Computer, Inc. *)
(* All Rights Reserved *)
(* ----- *)
(* ----- *)
(* Modification History *)
(* ----- *)
(* Date | By | Description *)
(* ----- *)
(* 1-Feb-88 | DK | file created *)
(* 13-May-88 | DK | Added user area to connection record*)
(* 13-May-88 | DK | Added File Transfer Record*)
(* ----- *)
(******)

```

UNIT adspcmd;

INTERFACE

USES Memtypes, QuickDraw, OSIntf, ToolIntf, AppleTalk, ADSP;

CONST

```

ASYNC      = TRUE;
SYNC       = FALSE;

(** connection mode status **)
NOP        = 0;
REQ        = 1;
ACK        = 2;
EST        = 3;

GLOBALNBPDATA = 28400;
GLOBALDSPDATA = 29337;
GLOBALSKTDATA = 21644;
DEFAULT_ERROR = 23220;
NO_ERROR     = 22476;
MEM_ERROR    = 31830;
NBPLSIZE     = 120;
ATPBSIZE     = 578; ( standard size of "transaction" buffer )

INTERVAL     = 20;
( default retry interval 60 ticks=10 secs.)
RETRY        = 3; (retry count = 3: total = 3 * 60 = 30 secs.)

PORTBUSE     = $291;
SPCONFIG     = $1FB;

CLOSE_OK     = 0;
RECEIVING    = 1;
CLOSE_NOW    = 2;

NEWLINE      = $0D;
( inserted after each entry in the zone info table)

aTalkVars    = $2D8;
( pointer to appletalk vars, aka aBusVars, mppVars)
sysABridge    = $19;
( Node address of a bridge [byte] )
sysNetNum     = $1A; ( This node's network number [word])

(***** File Transfer Protocol *****)
NO_FORK       = 0; ( Currently not sending any data)
DATA_FORK     = 1; ( Currently sending the data fork of file)
RESOURCE_FORK = 2; ( Currently sending resource fork of file)
FINDER_FORK   = 3; ( Currently sending finder bytes of file)

FILE_NOT_READY = 0; ( Not ready to send the file yet)
FILE_READY    = 1; ( ready to receive the file)

(******)
(* As long as you're asking... *)

```

```

(*)
(* The following data blocks reference memory pointers rather
than handles. This is *)
(* not an intentionally egregious use of the memory manager
but rather a way to *)
(* insure that all of the data in the connection block is
non-relocatable since *)
(* ADSP runs asynchronously, the data must always be presented
to ADSP. Hypercard *)
(* doesn't seem to mind non-rels and tends to do a fairly good
job of concatenating*)
(* them low in the heap (Hypercard is no slouch when it comes
to nonrels either, so *)
(* we're just piggybacking our pointers on top of the large
nonrel area at the start*)
(* of the Hypercard application heap)., a ne fait rien.*)
(******)

```

TYPE

```

LIntPtr      = ^LongInt;
IntPtr       = ^INTEGER;

CBPtr        = ^Connection;
Connection = Packed Record
  next: CBPtr; ( pointer to next block in the list)
  last: CBPtr; ( pointer to the last block in the list)
  ccbRef: Integer; ( reference number for this connection )
  mode: Integer;
  ( set to EST if the connection is open & ready)
  adr: AddrBlock;
  ( address of remote end (NIL if not connected))
  msg: Handle; ( callback message for incoming data)
  sendQ: Ptr;
  ( buffer for sending to remote connection end )
  recvQ: Ptr;
  ( buffer for receiving from remote connection)
  attn: Ptr; ( buffer for attention messages)
  outBuf: Handle; ( where the outgoing data is placed)
  inBuf: Handle; ( where the input data goes)
  attnBuf: Handle; ( where the attention data goes)
  remName: Handle; ( entity name of the remote end)
  ccb: TRCCB; ( pointer to connection control block)
  attnPB: DSPParamBlock;
  ( attention messages parameter block )
  dspPB: DSPParamBlock;
  ( connection param block for this connection)
  user: Handle; ( place for user defined data.)
End;

ADSPPtr      = ^ADSPBlock;
ADSPBlock = Packed Record ( ADSP protocol data)
  dspRefNum: Integer; ( driver refnum for ADSP )
  ccbref: Integer; ( ccbRefNum connection listener)
  addr: AddrBlock; ( socket address of this entity)
  ends: CBPtr; ( list of established connections)
  ccb: TRCCB; (ptr to listener connectn contrl blk)
  pb: DSPParamBlock;
  ( parameter block for connection listener)
  checkPoint: Integer; ( set if we're in a callback now)
  oldSelf: Byte; ( old state of the self-send flag)
  pad: Byte; ( keep em even, steven)
End;

NBPPtr      = ^NBPBlock;
NBPBlock = Packed Record
  Registered: Integer;
  ( true if we are already registered )
  EntCount: Integer; ( number of entities visible)
  LookUpBuffer: Handle; ( handle to the lookup buffer)
  NTEnt: NamesTableEntry; ( entry into names table)
  NBPLocal: array[1..NBPLSIZE] of char;
  ( used internally by NBP )
End;

```

```

FXPtr      = ^FXRecord;
FXRecord = Packed Record ( File Transfer Record.)
    Status : Integer;    ( state of the current connection)
    Fork    : Integer;    ( current for being transferred)
    FilePtr : Ptr;        ( Data buffer for the transfer)
    refNum  : Integer;    ( reference id to the file )
    Name    : Str255;     ( file name )
    io      : paramBlockRec; ( io parameter block for the i/o)
END;

```

End.

Listing 2:

```

(*****
(*file:   XCndGlue.Inc      *)
(*       *)
(* Support routines for ADSP *)
(* Hypercard interface.    *)
(*       *)
(* By:    Donald Koscheka   *)
(* Date:  9-Jan-89         *)
(* All Rights Reserved     *)
(*       *)
(*       *)
(*****

```

```

(*****
(*       *)
(* Miscellaneous support routines *)
(*       *)
(*****

```

```

FUNCTION RetrieveData( containerName : Str255 ): LongInt;
(*****
* Given the name of a Hypercard
* global container, return the
* data stored there as a long
* Integer.
*
* Container data is a handle
* to the contents of a global
* container. Theses contents
* should contain the string
* representation of a long integer
* which we convert to pascal
* longInt.
*
* Since containerData is a
* copy of the data, we must
* dispose of it afterwards.
*****
VAR
    containerData: Handle;
    longStr      : Str255;
BEGIN
    ContainerData := GetGlobal( containerName );

    (**If the container is not empty, convert it**)
    IF (ContainerData <> NIL) AND (ContainerData^ <> NIL) THEN
        BEGIN
            HLock( ContainerData );
            ZeroToPas( ContainerData^, longStr );
            RetrieveData := StrToLong( longStr );
            HUnlock( ContainerData );
            DisposHandle( ContainerData );
        END;
    END;

FUNCTION StringToShort( str : Ptr ): INTEGER;
(*****
* Given a pointer to a "C"
* string, convert the number

```

```

* to an integer and return
* it to the caller
*****
VAR
    tempStr : Str255;
BEGIN
    ZeroToPas( str, tempStr );
    StringToShort := Integer(StrToNum(tempStr));
END;

```

```

Function StrCmp( s1, s2 : String ): Boolean;
(*****
* Given two strings, return true
* if they match exactly, false
* otherwise
*****
VAR
    i,j      : Integer;
    matching : Boolean;
BEGIN
    matching := true;
    i := Integer( s1[0] ); (** number of characters in str **)
    j := Integer( s2[0] ); (** as well as comparison string **)
    IF i = j THEN (** if not same len, not same string **)
        WHILE i > 0 DO
            BEGIN
                WHILE matching DO
                    IF s1[i] <> s2[i] THEN (** strings don't match **)
                        BEGIN
                            matching := false;
                            i := 0;
                        END;
                    i := i - 1;
                END;
            StrCmp := matching;
        END;

```

```

PROCEDURE p2cStr( myPtr : Ptr );
(*****
* Convert a pascal string to a
* 'c' string in place
*****
VAR
    i : LongInt;
    aPtr: Ptr;
BEGIN
    FOR i := 0 to myPtr^ DO
        BEGIN
            aPtr := Pointer( ORD( myPtr ) + 1 );
            myPtr^ := aPtr^;
            myPtr := aPtr;
        END;
    myPtr := Pointer( ORD( myPtr ) - 1 );
    myPtr^ := 0;
END;

```

```

PROCEDURE TrashHandle( inH : Handle );
(*****
* Unlock the input handle
* and deallocate it
*****
BEGIN
    IF inH <> NIL THEN
        BEGIN
            HUnlock( inH );
            DisposHandle( inH );
        END;
    END;

```

```

(*****
(*       *)
(* Routines to access entities in the *)
(* names table *)

```

```

FUNCTION NBPGetAddress(name : Ptr) : AddrBlock;
(* ***** *)
* Given the name of an entity
* walk through the names table
* for the name and return
* the address if found, NIL
* otherwise
*
(* ***** *)
VAR
    error      : Integer;
    nbp        : NBPPtr;
    found      : Boolean;
    eName      : EntityName;
    eAddr      : AddrBlock;
    who        : Integer;
    str        : StringPtr;
    pName      : Str255;

BEGIN
    found := FALSE;
    who := 1;
    nbp := NBPPtr( RetrieveData( 'GLOBALNBPDATA' ) );
    IF ( nbp <> NIL ) AND ( nbp.LookUpBuffer <> NIL ) THEN
        BEGIN
            str := StringPtr( @eName.objStr );
            ZeroToPas( name, pName );
            NBPGetAddress.aNet := 0;
            NBPGetAddress.aNode := 0;
            NBPGetAddress.aSocket := 0;

            HLock( Handle(nbp.LookUpBuffer) );
            IF ( nbp.EntCount > 0 ) THEN
                REPEAT
                    error := NBPExtract( Ptr(nbp.LookUpBuffer),
                                         nbp.EntCount, who, eName, eAddr );
                    IF error = noErr THEN
                        BEGIN
                            found := StringEqual( str, pName );
                            IF found THEN
                                NBPGetAddress := eAddr
                            ELSE
                                who := who + 1;
                                IF ( who > nbp.EntCount ) THEN found := TRUE;
                            END
                        ELSE
                            found := TRUE;
                        UNTIL found;
                        HUnlock( Handle(nbp.LookUpBuffer) );
                    END;
                END;
            END;

FUNCTION NBPGetName( addr : AddrBlock ) : Handle;
(* ***** *)
* Given the address of an entity,
* walk through the names table
* searching for the address and
* return the name of the entity,
* nil if not
*
(* ***** *)
VAR
    error      : Integer;
    nbp        : NBPPtr;
    done       : Boolean;
    eName      : EntityName;
    eAddr      : AddrBlock;
    who        : Integer;
    yourName   : Handle;

BEGIN
    done := FALSE;
    who := 1;

```

```

    yourName := NIL;
    nbp := NBPPtr( RetrieveData( 'GLOBALNBPDATA' ) );

    IF ( nbp <> NIL ) AND ( nbp.LookUpBuffer <> NIL ) THEN
        BEGIN
            HLock( nbp.LookUpBuffer );
            IF ( nbp.EntCount > 0 ) THEN
                REPEAT
                    error := NBPExtract(
                        nbp.LookUpBuffer, nbp.EntCount, who, eName, eAddr );
                    IF error = noErr THEN
                        BEGIN
                            IF LongInt(addr) = LongInt(eAddr) THEN
                                yourName := PasToZero( eName.objStr )
                            ELSE
                                who := who + 1;

                                IF ( who > nbp.EntCount ) THEN
                                    done := TRUE;
                                END
                            ELSE
                                done := TRUE;
                            UNTIL ( done ) OR ( yourName <> NIL );

                            HUnlock( nbp.LookUpBuffer );
                        END;
                    NBPGetName := yourName;
                END;

                (* ***** *)
                (*
                 *
                 * Routines to hangup and tear down a
                 * connection
                 *)
                (* ***** *)

PROCEDURE DSPKillData( cb : CBPtr );
(* ***** *)
(* Either we could not allocate enough *)
(* memory or the connection would not *)
(* initialize, dump all allocated data *)
(*
   *)
(* The input parameter, cb, points to *)
(* the block that we want to kill
   *)
(*
   *)
(* ***** *)
BEGIN
    IF cb <> NIL THEN
        WITH cb DO
            BEGIN
                (** could not allocate or initialize **)
                IF sendQ <> NIL THEN DisposPtr( sendQ );
                IF recvQ <> NIL THEN DisposPtr( recvQ );
                IF attn <> NIL THEN DisposPtr( attn );

                TrashHandle( remName );
                TrashHandle( msg );
                TrashHandle( inBuf );
                TrashHandle( outBuf );
                TrashHandle( attnBuf );

                DisposPtr( Ptr( cb ) );
            END;
        END;

PROCEDURE DSPDisconnect( cn : CBPtr );
(* ***** *)
* Tear down an ADSP connection,
* deallocate the memory and remove
* the entity from the connection list.
(* ***** *)
VAR
    nb, lb : CBPtr;
    dpb : dspParamBlock;

```

```

tH      : Handle;

BEGIN
  IF cn <> NIL THEN
    BEGIN
      IF cn^.remName <> NIL THEN
        BEGIN
          SetGlobal( 'ADSPCaller', cn^.remName );
          SendCardMessage( 'HungUp' );
        END;

        (** remove connection from connection list**)
        IF cn = adsp^.ends THEN (* special case: cn = first *)
          BEGIN
            adsp^.ends := cn^.next;
            nb := cn^.next;
            IF nb <> NIL THEN nb^.last := NIL;
          END
        ELSE
          BEGIN
            nb := cn^.next;
            lb := cn^.last;
            IF nb <> NIL THEN nb^.last := lb;
            IF lb <> NIL THEN lb^.next := nb;
          END;

          DSPKillData( cn );
        END;
      END;
    END;

  FUNCTION DSPHangUp( cn : CBPtr ):OSErr;
  (*****
  * Hangup the line on requested
  * connection end
  *
  *
  * IN:
  * pointer to the connection's block
  * handle to the data to send
  *
  *****)
  VAR
    myPB : DSPParamBlock;

  BEGIN
    IF cn <> NIL THEN (** If connection exists, tear down**)
      BEGIN
        WITH myPB DO
          BEGIN
            ioCRefNum := adsp^.dspRefNum;
            ccbRefNum := cn^.ccbRef;
            csCode := dspRemove;
            abort := -1;

            (** Hangup synchronously because the state of the
            connection is about to go ballistic**)
          END;
          DSPHangup := PBControl( @myPB, SYNC );
          DSPDisconnect( cn );
        END;
      END;

      (*****
      (*
      (* Routines to install a connection *)
      (* and call a remote party. *)
      (*
      *****)

      PROCEDURE DSPAddConnection( cb : CBPtr );
      (*****
      * If we have a good connection, we can
      * go ahead and put it into the connection
      * list. It is either the first item in

```

```

* the list or we append it to the end by
* traversing the list to find the last item
*****
      VAR
        nb, lb : CBPtr;
      BEGIN
        nb := adsp^.ends;

        IF cb <> NIL THEN
          IF nb = NIL THEN
            adsp^.ends := cb;
          ELSE
            BEGIN
              WHILE nb <> NIL DO (** walk to end of list**)
                BEGIN
                  lb := nb;
                  nb := nb^.next;
                END;

                lb^.next := cb;
                cb^.next := NIL;
                cb^.last := lb;
              END;
            END;
          END;

        FUNCTION DSPCreate( addr:AddrBlock;
          inBufSiz,
          outBufSiz,
          retry,
          interval :Integer;
          message : Handle ): CBPtr;
        (*****
        * Creates an ADSP connection,
        * adds it to the connection list,
        *
        *
        * IN:
        * addr : address of the entity
        * inBufSiz,
        * outBufSiz
        *
        * : size of the buffers
        * retry : number of times to
        * retry the open
        * interval : 10Xticks time to wait
        * between retries
        * mess : callback message
        *
        * OUT:
        * CBPtr : return a pointer to
        * the connection, NIL if
        * connection not initied
        *
        *****)
        VAR
          error : Integer;
          siz : LongInt;
          cb : CBPtr;
          nbp : NBPPtr;
          haveIt : BOOLEAN;

        BEGIN
          cb := CBPtr( NewPtr( SizeOf( Connection ) ));
          IF cb <> NIL THEN
            WITH cb DO (** Allocate connection's structures**)
              BEGIN
                ccbRef := 0;
                msg := NIL;
                last := NIL;
                next := NIL;
                adr := addr; (** address of the remote end **)
                inBuf := NIL; (** no data received yet**)
                outBuf := NIL; (** no data sent yet **)
                attnBuf := NIL; (** no attention message...**)

```


HyperChat™

XCMD Corner: Window Scrollbars



HyperCard

Joe Zuffoletto
Apple Computer, Inc.
Volume 5, Number 3

In Last Month's Episode...

Last month I showed you how to write an event-driven XCMD that displays a normal Macintosh window on top of HyperCard. I discussed some interesting human interface issues that come up when you do this, and I demonstrated how to communicate with HyperCard from an XCMD in order to do window updates and other “human interface maintenance” tasks. I also used the XCMD as a springboard for illustrating more general techniques, including how to be MultiFinder aware, how to do cursor tracking under MultiFinder, and how to cope elegantly with multiple-screen environments. Finally, I gave the window scroll bars that didn't work, and I promised to “breathe life into them” this month, which I will do by adding code to the XCMD that takes a “snapshot” of the current card and displays it in the window. We can then scroll around and admire the snapshot until we either get bored, learn about scrolling, or both!

Scroll Bars!

The scroll bars are here!

The scroll bars are here!

Why should I care?

People have been scrolling for years....

Yes, but few people have taken the time and effort to *explain* scrolling to neophyte Mac programmers, who must either figure it out themselves, copy code from someone else, or avoid it completely. Why is this? I have a simple theory: People are ashamed of their scrolling code. In fact, some people might be *afraid* of their scrolling code. Scrolling code is ugly. It consists of gobs of source riddled with special cases and questionable algorithms. Nobody has found an elegant way to do it, so nobody wants to show their version of it to anybody else.

On top of all this, scrolling is implemented differently depending on the type of document being scrolled, be it bitmapped graphics, object-oriented graphics, text, icon lists, etc. While the basic rules stay the same, the special cases change and often multiply. Therefore, learning everything there is to know about scrolling is a daunting task indeed.

I certainly don't know everything about scrolling, but I know enough to get you started if you've never done it before, and I can touch on a few fundamentals that may help you improve your scrolling code if you've written some already. In this article I will show how to scroll bitmapped graphics documents, including how to set up and manage an offscreen bitmap for fast, flicker-free scrolling. I will assume we have a static document that cannot be edited or moved around inside its window. I leave these more challenging cases as proverbial exercises for the reader.

If you haven't done so already, read the Control Manager chapter of *Inside Macintosh*, Volume I so I don't have to explain

the Toolbox software behind controls. I will repeat some of the general material in that chapter, but I will not go into the data structures and other nuts-and-bolts information because there just isn't room for it here.

Those of you who have never written scrolling code stand to benefit the most because, like I said, annotated examples of it don't exactly abound. Unfortunately, the only thing worse than implementing scroll bars is explaining how the code works, so bear with me because there is a lot to cover and most of it is very dry. My plan is to first discuss the anatomy and arithmetic of scroll bars, then talk about all the special cases and implementation details. Finally, as a special bonus, I will describe how to set up and maintain an offscreen bitmap for a document you are scrolling.

This is a ton of material, but remember: No pain, no gain; no guts, no glory. Onward!

The Anatomy of Scroll Bars

Most of this section is a review of material in the Control Manager chapter of *Inside Macintosh*, Volume I, and of the Scroll Bars specification in *Apple's Human Interface Guidelines: The Apple Desktop Interface* (Addison-Wesley, 1987) I repeat it here in order to fix terms and to lend continuity to the article.

Scroll bars consist of five parts, as illustrated in Figure 1. These parts are the up arrow, page up region, thumb, page down region, and down arrow. The names are the same for a horizontal scroll bar, with the left end being “up.”

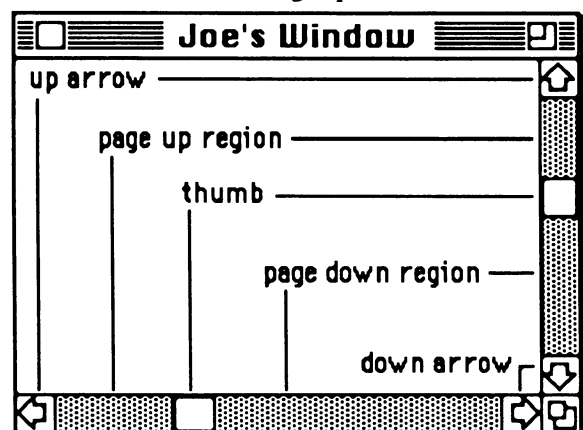


Figure 1

The up and down arrows scroll the document arbitrarily small but equal distances in their respective directions. If you are scrolling text, for example, the arrows might scroll it one line at a time. If you are scrolling graphics, you might shift your document 5 to 10 pixels at a time depending on how fast and

smooth you want the scrolling to be. In any event, the distance scrolled remains constant, regardless of window size.

Page up and page down scroll the document one “page” or “windowful” at a time. Most applications scroll slightly less than a windowful so users can keep track of where they are as they scroll through long documents. Obviously, the distance scrolled varies with window size.

Finally, the thumb can be dragged anywhere in the scroll bar; when it is released, the application scrolls to the corresponding position in the document. The distance scrolled depends only on the thumb displacement and is independent of window size.

Scroll bars can be either **active**, **unhighlighted**, or **inactive**. A scroll bar is active when the window that owns it is active and the document can be scrolled in the scroll bar’s direction; active scroll bars are illustrated in Figure 2a.

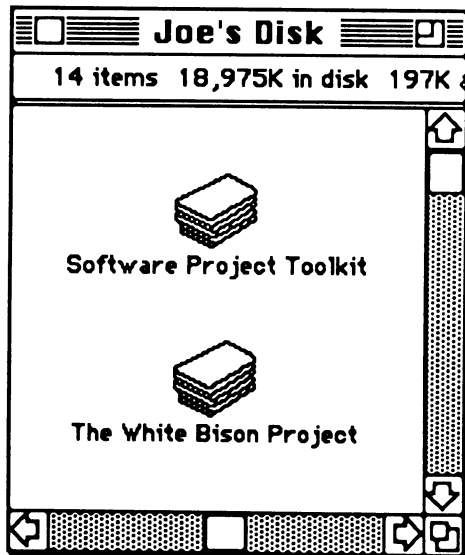


Figure 2a

If the owning window is active but the document cannot be scrolled (e.g., when the entire document is visible in the window), then the scroll bars are unhighlighted, and they appear as in Figure 2b.

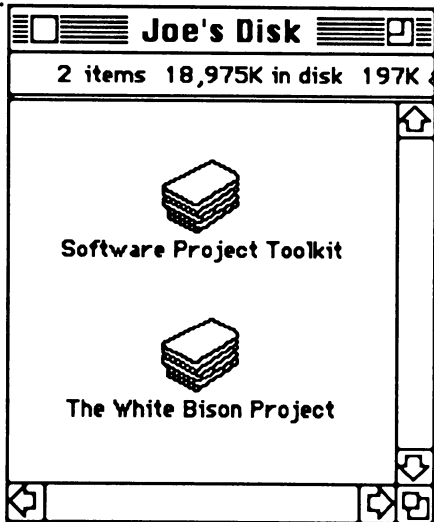


Figure 2b

Finally, if the owning window is inactive, the scroll bars must be inactive and look like those in Figure 2c.

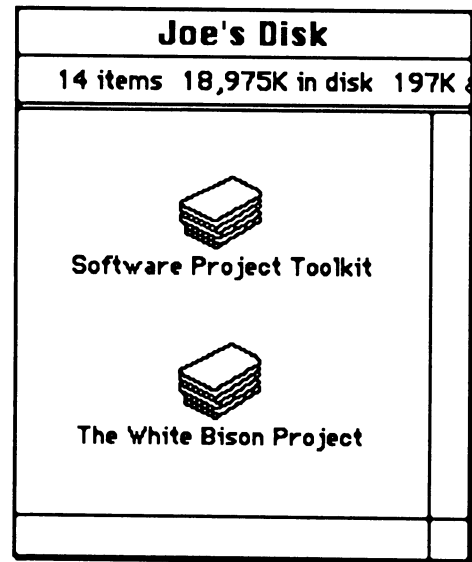


Figure 2c
Scroll Bar Arithmetic

You will find it much easier to write scrolling code if you can express scroll bar behavior in simple algebraic form. Good scrolling code, like QuickDraw, stands on a firm mathematical foundation. Building this foundation is best accomplished by drawing some pictures.

Figure 3 shows a large Macintosh window and a long, skinny document we wish to scroll. I have labeled important dimensions on the diagram; some of these will figure prominently in our algebraic expression of scrolling behavior.

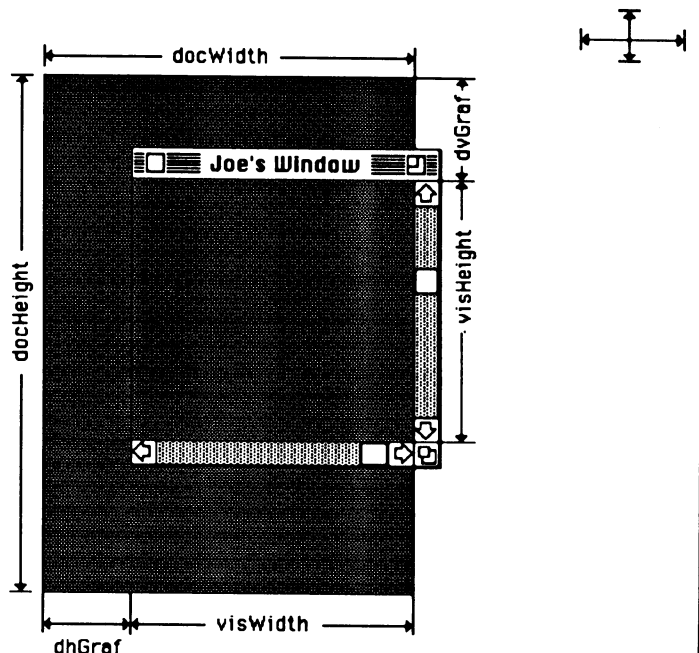


Figure 3

I define docWidth and docHeight as the width and height of

the document's rectangle; `visWidth` and `visHeight` as the width and height of the portion of the document visible in the window; and `dhGraf` and `dvGraf` as the distances the document has been scrolled in the horizontal and vertical directions, respectively. `dhGraf = dvGraf = 0` when the top left corners of the document and the window's `portRect` coincide, and both of these variables increase as the document moves up and to the left relative to the window; i.e., as we scroll down and to the right. The units for all these variables are pixels.

These are all the variables we need to express the document's position relative to the window. But we also need to express the thumbs' positions relative to the top (or bottom) of the scroll bars. The thumbs are moved by changing their *control values*; if we find the algebraic link between the control values and the variables described above, life will be great.

Inside Macintosh tells us that every control has predefined minimum and maximum values and a current value; this information is stored in the `ControlRecord` data structure for that control, and is initialized by us when we create the control with `NewControl`. The current value, of course, is where the thumb happens to be relative to the minimum and maximum values, so our first decision is: what should the minimum and maximum values be? Although you can choose a scheme as convoluted as you desire, I'll choose zero for the minimums, and `docWidth` and `docHeight` as the maximums of the horizontal and vertical scroll bars, respectively. I'll also initialize both current values (thumbs) to zero, placing us at the top left corner of the document.

The Golden Ratio of Scrolling

From now on, I'll simplify the discussion by considering only the vertical scroll bar; the exact same principles apply to the horizontal one. Here's a quick exercise: `dvGraf` limits how far we can scroll up and down in the document. It's smallest possible value is zero, when we are at the top of the document. What's the largest value `dvGraf` can have? `docHeight`? No! The correct answer is `docHeight - visHeight`, because once the bottom of the document is visible above the top of the horizontal scroll bar, we don't need to scroll down any more.

The largest value the *thumb* can have, however, is `docHeight`. At the top and bottom of the document, then, we have the following relationships:

Top: `dvGraf = 0`
`vThumb = 0`

Bottom: `dvGraf = docHeight - visHeight`
`vThumb = docHeight`

This suggests an important proportional relationship we must preserve as we scroll through the document:

$$vThumb = (dvGraf * docHeight) / (docHeight - visHeight)$$

As a check, substituting `dvGraf = 0` into the above equation gives `vThumb = 0`, and substituting `dvGraf = docHeight - visHeight` gives `vThumb = docHeight`. I call this the Golden Ratio of Scrolling, just because lots of high-falutin' proportions

in mathematics are called the Golden Ratio of something or another. My Golden Ratio will ensure that you scroll your documents the way users expect you to.

We now have enough mathematical background to implement scroll bars. Before diving into the code, however, let's learn about offscreen bitmaps and blitting, which will make our scrolling smooth and fast.

Bits, Blits, and CopyBits

You may have noticed that some applications have smooth, flicker-free scrolling while others have the choppy, flicker-ful variety. What you are witnessing is the almost total supremacy of `CopyBits` and `blitting` over `ScrollRect` in the scrolling arena. The basic idea is that `CopyBits` lets you blast an offscreen copy of your document to the screen immediately after you scroll (this is called blitting), while `ScrollRect` waits around for update events to be handled. On the downside, blitting takes a lot more memory, and you really have to be clever when you try to use it with large documents or large displays.

Scott Knaster gives a great explanation of blitting and offscreen bitmaps in his book *Macintosh Programming Secrets* (Addison-Wesley, 1988); I'm going to condense a similar discussion into a much smaller space here. If you get confused, check out pages 190-208 of Scott's book.

Let's take a concrete example. My `XCMD` takes a snapshot of HyperCard's card window, adds 16 pixels of white padding to the right and bottom edges, and stores the picture into an offscreen bitmap, which is just a nonrelocatable buffer in memory with a bitmap's structure. We will use `NewPtr` to allocate memory for the bitmap, so we need to determine how many bytes of memory we need. The card window is 342 pixels tall by 512 wide; adding 16 to both dimensions gives a rectangle 358 x 528 pixels. First we calculate `rowBytes` using Knaster's incredibly bizarre but effective formula:

```
rowBytes = (((pict.right - pict.left - 1) div 16) + 1) * 2
           = (((528 - 0 - 1) div 16) + 1) * 2
           = 66
```

From here on out things are much more intuitive:

```
bufferSize = (pict.bottom - pict.top) * rowBytes
            = 358 * 66
            = 23628 bytes
```

So to create our bitmap we make the following calls:

```
VAR myBits: BitMap;
```

```
myBits.rowBytes := 66;
myBits.baseAddr := NewPtr(23628);
( do error checking, of course )
SetRect(myBits.bounds, 0, 0, 528, 358);
```

All that's left is to copy a picture of the card window into the bitmap; see procedure `GetHCBitMap` in the source listing for details.

Scrolling and Blitting

Now that our offscreen bitmap is set up, it's time link it to the scroll bars. This is where that great eclectic concept of blitting comes in.

Quick and efficient blitting is not accomplished by blasting the entire offscreen image to the screen and clipping to the `portRect` (minus the scroll bars) of the target window. If you did

that and Bill Atkinson saw your code, he would say, "No, no, no, that's dumb."

Instead, Bill would suggest that you copy a piece of the offscreen image just big enough to fit in your window. Assuming you are scrolling around and want to see different pieces of the image, you need a scheme for deciding which piece to copy. This is where the concept of a **blit rectangle** comes in.

(By the way, Bill Atkinson probably knows how to compress an arbitrary 512 by 342 pixel bitmap into three bits, so he might suggest something even more efficient. If he does, listen carefully and write it all down, because you'll never figure it out yourself.)

A blit rectangle is an image of the onscreen window's portRect (minus the scroll bars) superimposed on the offscreen bitmap; see Figure 4. It is visWidth wide by visHeight tall. If you resize the onscreen window, you resize the blit rectangle accordingly, except that the blit rectangle can never be larger than the offscreen bitmap or you will blast the random garbage that lies beyond into your onscreen window. Similarly, when you scroll the onscreen window, you slide the blit rectangle the appropriate distance and direction, always staying within the bounding rectangle of the offscreen bitmap.

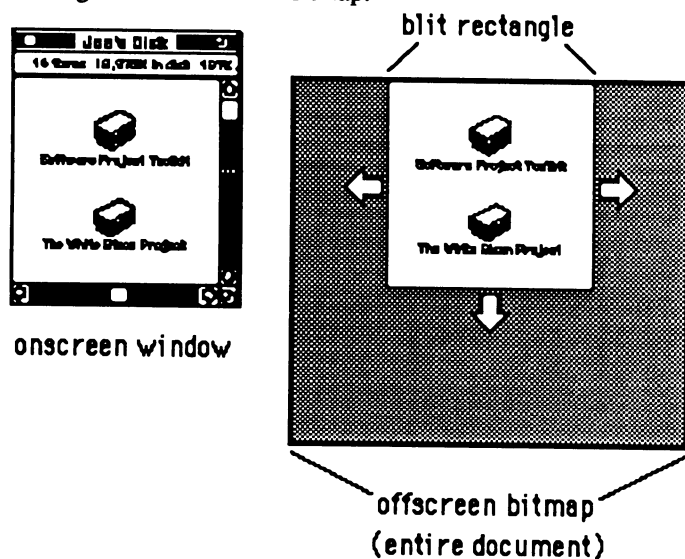


Figure 4

After you have resized or moved the blit rectangle as described above, you copy its contents to the onscreen window with CopyBits. In the case of scrolling, the copy is performed immediately, but in the cases of zooming or resizing, the copy is performed when update events are handled. Summarizing, the sequence of steps is:

- scroll onscreen window and move thumb
- slide blit rectangle proper distance and direction over offscreen bitmap (don't let it slide off, however!)
- copy contents of blit rectangle to onscreen window immediately
- or-
- zoom or resize onscreen window
- zoom or resize blit rectangle, up to size of offscreen bitmap

- copy contents of blit rectangle to onscreen window at update time

That's it for the concepts behind scrolling and blitting. Now we can look at the code and see how this stuff is done in practice, complete with refCon data structures, special cases, and other nasty ugliness.

First, a Pair of Data Structures

In this section and the ones that follow, I'm going to walk through the code that was stubbed out of last month's listing. For your convenience, however, I'm including the entire source of the Window XCMD.

Starting with the TYPE declarations, notice the two data structures I've defined, OffScrRecord and ScrollRecord (wonder no more, newcomers to Mac programming, you're about to find out what on earth refCon fields are used for!). OffScrRecord contains window-specific information, including a pointer to the grafPort containing the offscreen bitmap for the window, the width and height of the document in the bitmap, the current size of the blit rectangle, and current scroll state of the document. A handle to this information is stored in the window's refCon field, and I continually update the OffScrRecord via this handle in my scrolling code.

ScrollRecord contains two rather cryptic fields that I use for human interface purposes; I'll explain them later. In this program, I create only one ScrollRecord on the heap and share it among all my scroll bars. You'll soon understand why this is possible.

Allocating space on the heap for these data structures is accomplished with a parentheses-rich Pascal statement of the form:

```
myBlobHandle := BlobHandle(NewHandle(SizeOf(BlobRecord)));
```

How to Initialize Scroll Bars

Let's skip down to the main program, and I'll touch on the procedures used for initializing the scroll bars and offscreen bitmap.

After we've put the owning window on the heap, we go ahead and initialize its scroll bars. First we allocate space for the ScrollRecord, because if there's no room for it we have to quit. Assuming there is space, we grab our handle to it and initialize the ScrollRecord's fields. Again, I'll get to what they mean in a minute.

Next we call CreateHScrollBar and CreateVScrollBar, functions of my own design that allocate and return ControlHandles to the respective scroll bars for a given window. I recommend you name your controls as shown because it helps out in debugging.

HiliteScrollBars is called next; it checks the size of the window against the size of the document and unhighlights the appropriate scroll bar if the window is bigger in a given direction (see Figure 2b). Otherwise it checks the scrolled state of the document (given by dvGraf and dhGraf; see Figure 3) and positions the thumbs accordingly. Notice the use of the Golden Ratio. Notice also that HiliteScrollBars is called whenever:

- the window is made active

- the window is resized or zoomed
- the window is updated

After the window and scroll bars are drawn, DrawContents blasts the bits inside the blit rectangle to the screen. The offscreen bitmap has already been initialized by GetHCBiMap, and the blit rectangle has been initialized to the starting window size by the InitBlit procedure.

Scrolling

If we got this far, we should be zipping along in our main event loop, ready to handle events to our scroll bars. Remember from *Inside Macintosh* that the scroll bars sit on the content region of the window, so when we get a mouseDown there, we first call FindControl, passing it the location of the mouseDown in global coordinates, to find out which control, if any, was hit.

One possibility is that the user grabs the thumb, in which case we call ScrollWithThumb. This routine uses the control titles to determine which of the two thumbs, horizontal or vertical, was grabbed; then it records the start value of the thumb before the thumb is moved. At this point TrackControl is called to handle the dragging of the thumb; when the thumb is released, the ball is in our court again and we immediately record the thumb's end value. The difference between the start and end values (given by the LONGINT amountToScroll) is munged through a rearranged version of the Golden Ratio, and the result tells us how far to scroll the document. The actual scrolling is accomplished by ScrollContents, which first slides the blit rectangle to the proper location on the offscreen bitmap before calling DrawContents to blast the bits to the screen.

If the user clicks instead on one of the arrows or on a page up/page down region, things get a little more complicated. Let me begin by explaining a feature I call "accelerating thumbs."

Today's Macintoshes, of course, are much faster than their early predecessors, which is wonderful. Because of this, however, it is sometimes very difficult to use scroll bars with any precision. Ever try to scroll a list box one line at a time on a Mac II? It requires the reflexes of a fighter pilot to scroll less than a windowful when you hit the up or down arrows. To solve this annoying problem I invented two-speed accelerating thumbs, which scroll slowly at first but switch to top speed if you hold the mouse down long enough. The page up and page down regions work the same way.

Accelerating thumbs are the reason behind the ScrollRecord data structure. The fields of this data structure work in conjunction with procedure MyScroll, which is the actionProc TrackControl calls repeatedly when the user holds the mouse button down in the scroll arrows or page up/page down regions (see pp. 1323-324 of *Inside Macintosh*). Each time the user clicks in one of these areas, heldDown is initialized to zero and goFast to true before TrackControl is called. TrackControl in turn calls MyScroll, and calls it repeatedly until the mouse button is released. Each pass through MyScroll scrolls the document one notch, the size of the notch depending on whether an arrow or a page region is being pressed. It also increments heldDown by one and compares it to a preset switching value, which in this case is

2. While heldDown is less than 2 we scroll slowly, but when it is greater than 2, goFast is set to true and we switch into high gear. When the user releases the mouse button, TrackControl is finished and the process re-initialized next time a scroll bar is hit.

MyScroll is definitely the workhorse procedure for the "hard part" of scrolling, which is handling the arrows and page regions. I could discuss its workings in great detail here but that would be no more informative than reading the source comments. Therefore, I refer you to the source comments. Just be aware that MyScroll does the math and exception handling for the scroll bars, making extensive use of the Golden Ratio, then calls ScrollContents to adjust the blit rectangle accordingly. Finally, ScrollContents calls DrawContents to blit the stuff to the screen.

Notice that procedure MyScroll and the routines it calls are positioned outside of the Window procedure. This is because MyScroll is an actionProc, and the linker can't assign a procPtr to it if it is nested. This violates the typical layout of an XCMD but is perfectly legal, as long as you don't declare global variables.

Wrapping Up

If you study this article and the accompanying code listing carefully, you will gain a clear understanding of the basics of scrolling, offscreen bitmaps, and blitting. If you are a relative newcomer to Mac programming, you should congratulate yourself for learning one of the most challenging aspects of the Apple desktop interface in record time.

Scrolling has some more knots in it that I didn't even cover. For example, what if the user edits the document being scrolled, possibly changing its length or width? What if you are scrolling a combination of text and graphics? What if your document is too big to fit into an offscreen bitmap? And how about auto-scrolling (when the user drags beyond an edge of the window)? The list goes on and on.

None of these knots are remote possibilities; if your application is at all useful they are inevitable! But untangling them is much easier if you understand the basics I've covered in this article. Take a look at what other applications are doing. Then draw some pictures, remember the Golden Ratio, and everything will be all right.

Happy scrolling!

(*

© 1988, Apple Computer, Inc.
All rights reserved.

Window2.p: A HyperCard XCMD in MPW Pascal 2.0.2
by Joe Zuffoletto
Version 1.0, 29 June 1988

Form: Window title, top, left, bottom, right

Example: window "My Window", 50, 100, 300, 400

Notes: Window puts up a standard document window with scroll bars. The window can be dragged, resized, zoomed, and closed.

Command-W is supported for closing the window.

Command-spacebar toggles the menubar on and off, as in HyperCard. If you try to draw the window's title bar off the screen or under the menubar, Window will abort with an error message. Error messages can be examined by looking at HyperCard's global variable "the result" after calling Window.

Window is MultiFinder friendly and works with HyperCard 1.2 or later. It supports multiple displays on the Mac II as well.

Window takes a snapshot of the HyperCard card window and displays it like a MacPaint document. You can scroll up and down, etc. This is just for demonstration and amusement. You must supply your own code for displaying whatever you want to display in the window.

To compile and link this file using MPW Pascal 2.0.2, select the following lines and press ENTER:

```
Pascal Window2.p
link -o "Hard Disk":HyperCard:"HyperCard Stacks":Home 0
    -rt XCMD=2000 -sn Main=Window 0
    Window2.p.o (MPW)Libraries:Interface.o 0
    (MPW)PLibraries:PasLib.o 0
    -m ENTRYPOINT
```

Use other link files as necessary.

The above link directives install the XCMD resource into the Home stack. You can substitute the name of any stack you want; be sure to provide the correct pathname. Also, make sure the target stack already has a resource fork or it won't work. You can create an empty resource fork in a stack with ResEdit.

*)

(\$R-)

(\$S Window)

UNIT DummyUnit;

INTERFACE

USES

MemTypes, QuickDraw, OSIntf, ToolIntf, PasLibIntf, HyperXCMD;

PROCEDURE EntryPoint(paramPtr:XCMDPtr);

IMPLEMENTATION

TYPE Str31 = String[31];

(Attach to refCon of a window)

OffScrHandle = ^OffScrRecPtr;

OffScrRecPtr = ^OffScrRecord;

OffScrRecord = RECORD

gPort: GrafPtr; (offscreen grafPort)
docWidth: INTEGER;
docHeight: INTEGER;
blitRect: Rect; (blit rectangle)
blitWidth: INTEGER;
blitHeight: INTEGER;
dhGraf: LONGINT;
dvGraf: LONGINT;

END;

(Attach to refCon of a scroll bar)

ScrollPtr = ^ScrollRecord;

ScrollPtr = ^ScrollRecord;

ScrollRecord = RECORD

heldDown: INTEGER; (see MyScroll)

goFast: BOOLEAN;

END;

PROCEDURE Window(paramPtr:XCMDPtr); FORWARD;

PROCEDURE EntryPoint(paramPtr:XCMDPtr);

BEGIN

Window(paramPtr);

END;

FUNCTION Min(int1,int2:INTEGER): INTEGER;

(Return the smaller of two integers.)

BEGIN

IF (int1 <= int2) THEN

Min := int1

ELSE

Min := int2;

END; (Min)

PROCEDURE InitBlit(theWindow:WindowPtr);

(Initialize the blit rectangle so it is in the upper left corner of the offscreen bitmap and so it is the same size as the target window onscreen.)

VAR

myOffScrHandle: OffScrHandle;

BEGIN

myOffScrHandle := OffScrHandle(GetWRefCon(theWindow));

MoveHHI(Handle(myOffScrHandle));

HLock(Handle(myOffScrHandle));

WITH myOffScrHandle^^ DO

BEGIN

dhGraf := 0;

dvGraf := 0;

(Make sure blit rectangle doesn't hang off edges of offscreen bitmap, just in case window is larger.)

blitWidth := Min((theWindow^.portRect.right - 15 -
theWindow^.portRect.left), docWidth);

blitHeight := Min((theWindow^.portRect.bottom - 15 -
theWindow^.portRect.top), docHeight);

SetRect(blitRect, gPort^.portBits.bounds.left,
gPort^.portBits.bounds.top,
gPort^.portBits.bounds.left + blitWidth,
gPort^.portBits.bounds.top + blitHeight);

END;

HUnlock(Handle(myOffScrHandle));

END; (InitBlit)

PROCEDURE InvalContents(theWindow:WindowPtr;
theOldSize:Rect);

(Perform "intelligent" window updating; i.e., if the window is grown, invalidate for the update event only the part that wasn't visible before.)

VAR

myOffScrHandle: OffScrHandle;

myWindowRect, windowRect: Rect;

tallRect, wideRect: Rect;

hWhiteSpace, vWhiteSpace: INTEGER;

hBlitOffset, vBlitOffset: INTEGER;

myDocWidth, myDocHeight: INTEGER;

newDhGraf, newDvGraf: LONGINT;

tempDH, tempDV: LONGINT;

BEGIN

hBlitOffset := 0;

vBlitOffset := 0;

myWindowRect := theWindow^.portRect;

myOffScrHandle := OffScrHandle(GetWRefCon(theWindow));

MoveHHI(Handle(myOffScrHandle));

HLock(Handle(myOffScrHandle));

WITH myOffScrHandle^^ DO

```

BEGIN
    tempDH := dhGraf;
    tempDV := dvGraf;
    myDocWidth := docWidth;
    myDocHeight := docHeight;
END;

(If we have scrolled to the bottom and/or right edges
of the document and then grow the window, the document
will "follow" the window so we don't just expose a lot
of white space. The same thing will happen if we zoom the
window to a larger size. In either case, we need to
invalidate the entire content region.)

(Is any white space being exposed in horizontal
direction?)

hWhiteSpace := (myWindowRect.right -
    myWindowRect.left - 15) -
    (myDocWidth - tempDH);

IF (hWhiteSpace > 0) THEN
BEGIN
    (If so, how much?)

    newDhGraf := tempDH - hWhiteSpace;

    (Move the document that distance, but not beyond the
    point where its left edge is flush with the left edge
    of the window.)

    IF newDhGraf < 0 THEN
        newDhGraf := 0;

    (Adjust blit rectangle accordingly.)

    hBlitOffset := INTEGER(tempDH - newDhGraf);
    tempDH := newDhGraf;
END; (IF hWhiteSpace)

(Repeat for vertical direction.)

vWhiteSpace := (myWindowRect.bottom -
    myWindowRect.top - 15) -
    (myDocHeight - tempDV);

IF (vWhiteSpace > 0) THEN
BEGIN
    newDvGraf := tempDV - vWhiteSpace;
    IF newDvGraf < 0 THEN
        newDvGraf := 0;
    vBlitOffset := INTEGER(tempDV - newDvGraf);
    tempDV := newDvGraf;
END; (IF vWhiteSpace)

(Save what we've learned to the data structure.)

WITH myOffScrHandle^^ DO
BEGIN
    dhGraf := tempDH;
    dvGraf := tempDV;
    blitWidth := Min((myWindowRect.right - myWindowRect.left -
15),
        (myDocWidth - dhGraf));
    blitHeight := Min((myWindowRect.bottom - myWindowRect.top
- 15),
        (myDocHeight - dvGraf));
    SetRect(blitRect, blitRect.left - hBlitOffset,
        blitRect.top - vBlitOffset,
        blitRect.left + blitWidth - hBlitOffset,
        blitRect.top + blitHeight - vBlitOffset);
    blitWidth := blitRect.right - blitRect.left;
    blitHeight := blitRect.bottom - blitRect.top;
END;
HUnlock(Handle(myOffScrHandle));

```

(If the document has followed the window, invalidate entire content region.)

```

IF ((hBlitOffset <> 0) OR (vBlitOffset <> 0)) THEN
BEGIN

```

```

    EraseRect(theWindow^.portRect);
    InvalRect(theWindow^.portRect);

```

```

END
ELSE

```

(Otherwise invalidate only the new white space.)

```

BEGIN

```

```

    SetRect(tallRect, theOldSize.right,
        theWindow^.portRect.top,
        theWindow^.portRect.right,
        theWindow^.portRect.bottom);

    SetRect(wideRect, theWindow^.portRect.left,
        theOldSize.bottom,
        theOldSize.right,
        theWindow^.portRect.bottom);

```

```

    EraseRect(tallRect);
    EraseRect(wideRect);
    InvalRect(tallRect);
    InvalRect(wideRect);

```

```

END; (IF hBlitOffset)
END; (InvalContents)

```

```

PROCEDURE DrawContents(theWindow: WindowPtr);
(Blasts contents of updated blit rectangle to screen.)

```

```

VAR

```

```

    myWindowRect, windowRect: Rect;
    myOffScrHandle: OffScrHandle;

```

```

BEGIN

```

```

    myOffScrHandle := OffScrHandle(GetWRefCon(theWindow));
    myWindowRect := theWindow^.portRect;
    MoveHHi(Handle(myOffScrHandle));
    HLock(Handle(myOffScrHandle));
    SetRect(windowRect, myWindowRect.left,
        myWindowRect.top,
        myWindowRect.left +
myOffScrHandle^^.blitWidth,
        myWindowRect.top +
myOffScrHandle^^.blitHeight);
    ClipRect(windowRect);
    CopyBits(myOffScrHandle^^.gPort^.portBits, theWindow^.portBits,
        myOffScrHandle^^.blitRect, windowRect, srcCopy, NIL);
    HUnlock(Handle(myOffScrHandle));
END; (DrawContents)

```

```

PROCEDURE ScrollContents(theWindow: WindowPtr;
    dh, dv: INTEGER);

```

```

VAR

```

```

    myOffScrHandle: OffScrHandle;
    leftSpace, topSpace: INTEGER;
    rightSpace, bottomSpace: INTEGER;

```

```

BEGIN

```

```

    myOffScrHandle := OffScrHandle(GetWRefCon(theWindow));
    MoveHHi(Handle(myOffScrHandle));
    HLock(Handle(myOffScrHandle));

```

(Calculate space between edges of blitRect and edges of the bitmap.)

```

WITH myOffScrHandle^^ DO

```

```

BEGIN

```

```

    leftSpace := gPort^.portBits.bounds.left - blitRect.left;
    topSpace := gPort^.portBits.bounds.top - blitRect.top;
    rightSpace := gPort^.portBits.bounds.right -
blitRect.right;
    bottomSpace := gPort^.portBits.bounds.bottom -
blitRect.bottom;

```

```

END;

```

(Then move blitRect, but not past the edge of the bitmap.)

```

IF (dv = 0) THEN
  IF (dh > 0) THEN (moving to the right)
    IF (rightSpace > dh) THEN
      OffsetRect(myOffScrHandle^.blitRect,dh,dv)
    ELSE
      BEGIN
        dh := rightSpace;
        OffsetRect(myOffScrHandle^.blitRect,dh,dv);
      END
    ELSE IF (leftSpace > dh) THEN (moving to the left)
      BEGIN
        dh := leftSpace;
        OffsetRect(myOffScrHandle^.blitRect,dh,dv);
      END
    ELSE
      OffsetRect(myOffScrHandle^.blitRect,dh,dv)
    ELSE IF (dv > 0) THEN (moving down)
      IF (bottomSpace > dv) THEN
        OffsetRect(myOffScrHandle^.blitRect,dh,dv)
      ELSE
        BEGIN
          dv := bottomSpace;
          OffsetRect(myOffScrHandle^.blitRect,dh,dv);
        END
      ELSE IF (topSpace > dv) THEN (moving up)
        BEGIN
          dv := topSpace;
          OffsetRect(myOffScrHandle^.blitRect,dh,dv);
        END
      ELSE
        OffsetRect(myOffScrHandle^.blitRect,dh,dv);

```

(Save actual distances moved to data structure.)

```

WITH myOffScrHandle^ DO
  BEGIN
    dhGraf := dhGraf + dh;
    dvGraf := dvGraf + dv;
  END;
  HUnlock(Handle(myOffScrHandle));

```

(Shoot new contents of blit rectangle to screen.)

```

  DrawContents(theWindow);
END; (ScrollContents)

```

```

PROCEDURE MyScroll(theControl:ControlHandle;
  partCode:INTEGER);

```

```

VAR
  myOffScrHandle: OffScrHandle;
  myScrollHandle: ScrollHandle;
  dh,dv: LONGINT;
  windowFull: LONGINT;
  myCtlValue: LONGINT;
  visHeight,visWidth: LONGINT;
  dontCare: LONGINT;
  tempDH,tempDV: LONGINT;
  startValue: INTEGER;
  myDocWidth,myDocHeight: INTEGER;
  direction: Str255;
  theWindow: WindowPtr;
BEGIN
  theWindow := theControl^.controlOwner;
  myOffScrHandle := OffScrHandle(GetWRefCon(theWindow));
  MoveHHi(Handle(myOffScrHandle));
  HLock(Handle(myOffScrHandle));

```

(Make local copies of data structure items.)

```

WITH myOffScrHandle^ DO
  BEGIN

```

```

  tempDH := dhGraf;
  tempDV := dvGraf;
  myDocWidth := docWidth;
  myDocHeight := docHeight;
END;
  HUnlock(Handle(myOffScrHandle));

```

(Initialize more variables for our formulas.)

```

  visHeight := theWindow^.portRect.bottom - 15 -
    theWindow^.portRect.top;
  visWidth := theWindow^.portRect.right - 15 -
    theWindow^.portRect.left;
  startValue := GetCtlValue(theControl);
  GetCTitle(theControl,direction);
  myScrollHandle := ScrollHandle(GetCRefCon(theControl));
  MoveHHi(Handle(myScrollHandle));
  HLock(Handle(myScrollHandle));

```

(Implementation of two-speed, accelerating scroll bars.)

```

WITH myScrollHandle^ DO
  BEGIN
    heldDown := heldDown + 1;
    IF (heldDown > 2) THEN
      goFast := TRUE;
  END; (WITH)

```

CASE partCode **OF**

```

  inUpButton:
    BEGIN
      (Don't scroll up if already at top!)
      IF (startValue > 0) THEN
        BEGIN
          IF (direction = 'MyVert') THEN
            BEGIN
              IF (visHeight >= myDocHeight) THEN
                BEGIN
                  setCtlValue(theControl,0);
                  ScrollContents(theWindow,0,-tempDV);
                END
              ELSE
                BEGIN
                  (Scroll up five pixels)
                  dv := tempDV - 5;

                  (Set new thumb value using Golden Ratio.)

                  myCtlValue := LONGINT((dv * myDocHeight) DIV
                    (myDocHeight - visHeight));
                  ClipRect(theWindow^.portRect);
                  setCtlValue(theControl,INTEGER(myCtlValue));

                  (Update blit rectangle, blast document to screen.)

                  ScrollContents(theWindow,0,-5);
                  END; (IF visHeight...ELSE)
                END
              ELSE
                BEGIN
                  (Repeat for horizontal direction.)
                  IF (visWidth >= myDocWidth) THEN
                    BEGIN
                      setCtlValue(theControl,0);
                      ScrollContents(theWindow,-tempDH,0);
                    END
                  ELSE
                    BEGIN
                      dh := tempDH - 5;
                      myCtlValue := LONGINT((dh * myDocWidth) DIV
                        (myDocWidth - visWidth));
                      ClipRect(theWindow^.portRect);
                      setCtlValue(theControl,INTEGER(myCtlValue));
                      ScrollContents(theWindow,-5,0);

```



```

hasWaitNextEvent:      BOOLEAN;
inBackGround,smallScreen:  BOOLEAN;
DoneFlag,HaveEvent:     BOOLEAN;
wTop,wLeft,wBottom,wRight:  INTEGER;
partCode,controlCode:    INTEGER;
largestHeight,largestWidth:  INTEGER;
dummy,charCode:          INTEGER;
screenWidth,screenHeight:  INTEGER;
myDocWidth,myDocHeight:    INTEGER;
eventPoint:              Point;
wRect,screenRect,dragRect: Rect;
winSizeLimits:            Rect;
oldSize:                  Rect;
newSize,dontCare:         LONGINT;
envError:                  OSerr;
cursorRgn:                RgnHandle;
hScroll,vScroll:          ControlHandle;
whichControl:              ControlHandle;
myWindow,whichWindow:      WindowPtr;
fatBitsWindow:             WindowPtr;
HCrefresh:                Str31;
wT,wL,wB,wR,wTitle:       Str255;
toolStr,patStr,msgStr:     Str255;
widthStr,heightStr:        Str255;
myBits:                    BitMap;
theEnv:                    SysEnvRec;
myEvent:                   EventRecord;
wRecord:                   WindowRecord;
HCPort:                    GrafPtr;
theOffScrHandle:           OffScrHandle;
theScrollHandle:           ScrollHandle;
myOffScr:                  OffScrRecord;
myScrollRecord:            ScrollRecord;

```

```

FUNCTION TrapAvailable(tNumber: INTEGER; tType:
                        TrapType): BOOLEAN;
(Check to see if a given trap is implemented.)
BEGIN
    TrapAvailable := NGetTrapAddress(tNumber, tType) <>
                    GetTrapAddress(Unimplemented);
END; (TrapAvailable)

```

```

FUNCTION CreateHScrollBar(theWindow: WindowPtr;
                        theValue,theMin,theMax: INTEGER;
                        theRefCon: LONGINT): ControlHandle;
(Allocate and draw a horizontal scroll bar in the window.
Return a controlHandle to the scroll bar.)
VAR
    myWindowRect,hScrRect: Rect;
BEGIN
    SetPort(theWindow);
    myWindowRect := theWindow^.portRect;
    SetRect(hScrRect,myWindowRect.left - 1,
            myWindowRect.bottom - 15,
            myWindowRect.right - 14,
            myWindowRect.bottom + 1);
    CreateHScrollBar := NewControl(theWindow,hScrRect,
                                'MyHoriz',TRUE,
                                theValue,theMin,theMax,
                                scrollbarProc,theRefCon);
END; (CreateHScrollBar)

```

```

FUNCTION CreateVScrollBar(theWindow: WindowPtr;
                        theValue,theMin,theMax: INTEGER;
                        theRefCon: LONGINT): ControlHandle;
(Allocate and draw a vertical scroll bar in the window.
Return a controlHandle to the scroll bar.)
VAR
    myWindowRect,vScrRect: Rect;
BEGIN
    SetPort(theWindow);
    myWindowRect := theWindow^.portRect;
    SetRect(vScrRect,myWindowRect.right - 15,
            myWindowRect.top - 1,

```

```

            myWindowRect.right + 1,
            myWindowRect.bottom - 14);
    CreateVScrollBar := NewControl(theWindow,vScrRect,
                                'MyVert',TRUE,
                                theValue,theMin,theMax,
                                scrollbarProc,theRefCon);
END; (CreateVScrollBar)

```

```

PROCEDURE InvalScroll(theWindow: WindowPtr);
(Accumulate the rectangles occupied by the window's
horizontal and vertical scroll bars into the update
region.)
VAR
    theRect,tallRect,wideRect: Rect;
BEGIN
    SetPort(theWindow);
    theRect := theWindow^.portRect;
    ClipRect(theRect);

    (Accumulate tallRect, which is occupied by the vertical
    scroll bar )

    SetRect(tallRect,theRect.right-15,
            theRect.top,
            theRect.right,
            theRect.bottom);

    EraseRect(tallRect);
    InvalRect(tallRect);

    (Accumulate wideRect, which is occupied by the
    horizontal scroll bar )

    SetRect(wideRect,theRect.left,
            theRect.bottom-15,
            theRect.right,
            theRect.bottom);

    EraseRect(wideRect);
    InvalRect(wideRect);
END; (InvalScroll)

```

```

PROCEDURE Deactivate(theWindow: WindowPtr);
(Deactivate the scroll bars in the window in accordance
with the human interface guidelines. This means we
must erase everything enclosed by the control rec-
tangles.)
VAR
    theControl: ControlHandle;
    theControlRect: Rect;
BEGIN
    (I always title my scroll bars 'MyVert' and 'MyHoriz'
    so I can easily find them by walking a window's
    control list.)
    theControl := WindowPeek(theWindow)^(theWindow^.controlList);
    WHILE (theControl <> NIL) DO
        BEGIN
            IF (theControl^.ctrlTitle = 'MyVert') OR
                (theControl^.ctrlTitle = 'MyHoriz') THEN
                BEGIN
                    theControlRect := theControl^.ctrlRect;
                    InsetRect(theControlRect,1,1);
                    EraseRect(theControlRect);
                END; (IF)
                theControl := theControl^.nextControl;
            END; (WHILE)
    END; (Deactivate)

```

```

PROCEDURE HiliteScrollBars(theWindow: WindowPtr);
VAR
    myWindowRect: Rect;
    visWidth,visHeight: INTEGER;
    myDocWidth,myDocHeight: INTEGER;
    myCtlValue: INTEGER;
    tempDH,tempDV: LONGINT;
    myOffScrHandle: OffScrHandle;

```



```

BEGIN
myOffScrHandle := OffScrHandle(GetWRefCon(theWindow));
MoveHHi(Handle(myOffScrHandle));
HLock(Handle(myOffScrHandle));

(Make local copies of data structure items.)

WITH myOffScrHandle^^ DO
BEGIN
tempDH := dhGraf;
tempDV := dvGraf;
myDocWidth := docWidth;
myDocHeight := docHeight;
END;

HUnlock(Handle(myOffScrHandle));
myWindowRect := theWindow^.portRect;
visHeight := myWindowRect.bottom - 15 - myWindowRect.top;
visWidth := myWindowRect.right - 15 - myWindowRect.left;
SetPort(theWindow);
ClipRect(myWindowRect);

(Check to see if window is taller than contents. If
so, unhilite vertical scroll bar. Otherwise, use
Golden Ratio to draw vertical scroll bar with thumb
in right position.)

IF ((visHeight >= myDocHeight) AND (tempDV = 0)) THEN
HiliteControl(vScroll, INTEGER(inactive))
ELSE
BEGIN
HiliteControl(vScroll, INTEGER(active));
IF (visHeight >= myDocHeight) THEN
SetCtlValue(vScroll, myDocHeight)
ELSE
BEGIN
myCtlValue := LONGINT((tempDV * myDocHeight) DIV
(myDocHeight - visHeight));
ClipRect(theWindow^.portRect);
setCtlValue(vScroll, INTEGER(myCtlValue));
END; (IF visHeight...ELSE)
END;

(Repeat for horizontal scroll bar.)

IF ((visWidth >= myDocWidth) AND (tempDH = 0)) THEN
HiliteControl(hScroll, INTEGER(inactive))
ELSE
BEGIN
HiliteControl(hScroll, INTEGER(active));
IF (visWidth >= myDocWidth) THEN
SetCtlValue(hScroll, myDocWidth)
ELSE
BEGIN
myCtlValue := LONGINT((tempDH * myDocWidth) DIV
(myDocWidth - visWidth));
ClipRect(theWindow^.portRect);
setCtlValue(hScroll, INTEGER(myCtlValue));
END; (IF visWidth...ELSE)
END; (IF visWidth...ELSE)

END; (HiliteScrollBars)

PROCEDURE MoveScrollBars(theWindow:WindowPtr);
(Call this procedure after theWindow has changed size.
MoveScrollBars erases theWindow's scroll bars, resizes
them, and redraws them.)
VAR
myWindowRect: Rect;
vScrRect, hScrRect: Rect;
BEGIN
myWindowRect := theWindow^.portRect;
SetRect(hScrRect, myWindowRect.left - 1,
myWindowRect.bottom - 15,

```

```

myWindowRect.right - 14,
myWindowRect.bottom + 1);
SetRect(vScrRect, myWindowRect.right - 15,
myWindowRect.top - 1,
myWindowRect.right + 1,
myWindowRect.bottom - 14);
SetPort(theWindow);
ClipRect(myWindowRect);

(Hide and resize the scroll bars to fit the new window
size.)

HideControl(hScroll);
HideControl(vScroll);

MoveControl(hScroll, hScrRect.left, hScrRect.top);
SizeControl(hScroll, (hScrRect.right - hScrRect.left),
(hScrRect.bottom - hScrRect.top));

MoveControl(vScroll, vScrRect.left, vScrRect.top);
SizeControl(vScroll, (vScrRect.right - vScrRect.left),
(vScrRect.bottom - vScrRect.top));

HiliteScrollBars(theWindow);
ShowControl(hScroll);
ShowControl(vScroll);
END; (MoveScrollBars)

PROCEDURE ScrollWithThumb(theControl:ControlHandle;
theEventPoint:Point);
VAR
theWindow: WindowPtr;
myOffScrHandle: OffScrHandle;
visWidth, visHeight: INTEGER;
myDocWidth, myDocHeight: INTEGER;
startValue, endValue: INTEGER;
dummy: INTEGER;
amountToScroll: LONGINT;
tempDH, tempDV: LONGINT;
direction: Str255;
BEGIN
theWindow := theControl^.controlOwner;
myOffScrHandle := OffScrHandle(GetWRefCon(theWindow));
MoveHHi(Handle(myOffScrHandle));
HLock(Handle(myOffScrHandle));

(Make local copies of the data structure items.)

WITH myOffScrHandle^^ DO
BEGIN
tempDH := dhGraf;
tempDV := dvGraf;
myDocWidth := docWidth;
myDocHeight := docHeight;
END;

HUnlock(Handle(myOffScrHandle));
GetCTitle(theControl, direction);
visHeight := theWindow^.portRect.bottom -
theWindow^.portRect.top - 15;
visWidth := theWindow^.portRect.right -
theWindow^.portRect.left - 15;

(Record where thumb is before it is moved.)

startValue := GetCtlValue(theControl);

(Move it.)

dummy := TrackControl(theControl, theEventPoint, NIL);

(Record where thumb is released.)

endValue := GetCtlValue(theControl);

(Record the distance thumb was moved.)

```

```

amountToScroll := LONGINT(endValue - startValue);

IF (direction = 'MyVert') THEN
BEGIN (moved vertical scroll bar)
  IF (endValue = 0) THEN (moved thumb to top)
    ScrollContents(theWindow,0,-tempDV)
  ELSE IF (endValue = myDocHeight) THEN
    (moved thumb to bottom)
    ScrollContents(theWindow,0,myDocHeight - tempDV)
  ELSE
    BEGIN
      (moved thumb somewhere in middle - use
      Golden Ratio to translate thumb movement
      into document movement.)
      amountToScroll := LONGINT((amountToScroll *
                                (myDocHeight -
                                visHeight)) DIV
                                myDocHeight);
    IF (ABS(amountToScroll) < 1) THEN
      (didn't move thumb far enough to scroll -
      snap it back to starting position.)
      SetCtlValue(theControl,startValue)
    ELSE
      ScrollContents(theWindow,0,
                    INTEGER(amountToScroll));
    END;
  END
ELSE (moved horizontal scroll bar)
BEGIN
  IF (endValue = 0) THEN
    ScrollContents(theWindow,-tempDH,0)
  ELSE IF (endValue = myDocWidth) THEN
    ScrollContents(theWindow,myDocWidth - tempDH,0)
  ELSE
    BEGIN
      amountToScroll := LONGINT((amountToScroll *
                                (myDocWidth -
                                visWidth)) DIV
                                myDocWidth);
    IF (ABS(amountToScroll) < 1) THEN
      SetCtlValue(theControl,startValue)
    ELSE
      ScrollContents(theWindow,
                    INTEGER(amountToScroll),0);
    END;
  END;
END; (ScrollWithThumb)

FUNCTION WhichDevice(thePoint:Point):GDHandle;
(For machines that support color QuickDraw and
multiple screens, WhichDevice figures out which screen
thePoint is on and returns a GDHandle to that screen.
thePoint might be where the mouse was clicked, for
example. Thanks to Greg Marriott for this code.)
VAR
  aDevice: GDHandle;
  foundOne: BOOLEAN;
BEGIN
  aDevice := GetDeviceList;
  foundOne := FALSE;

  (Walk the device list until thePoint is contained
  in some device's screen rectangle.)

  WHILE (aDevice <> NIL) AND NOT foundOne DO
    BEGIN
      IF PtInRect(thePoint,aDevice^.gdRect) THEN
        BEGIN
          WhichDevice := aDevice;
          foundOne := TRUE;
        END;
      aDevice := aDevice^.gdNextGD;
    END;
END;

```

```

END; (WhichDevice)

```

```

FUNCTION MenuBarHeight: INTEGER;
(Returns the height of the menubar in pixels, as read
from the low memory global mBarHeight.)

```

```

CONST
  mBarHeight = $BAA;
VAR
  menuBarHeightPtr: ^INTEGER;
BEGIN
  menuBarHeightPtr := Pointer(mBarHeight);
  MenuBarHeight := menuBarHeightPtr^;
END; (MenuBarHeight)

```

```

FUNCTION OnAScreen(theRect:Rect):BOOLEAN;
(OnAScreen returns FALSE if all of a window's title
bar is off the screen or if any part of it is under
the menubar. The portRect of the window to be checked
should be passed in theRect.)

```

```

CONST
  titleBarHeight = 18;
VAR
  deskRgn: RgnHandle;
  topLeft,topRight: Point;
BEGIN
  deskRgn := GetGrayRgn;
  topLeft.v := theRect.top - titleBarHeight;
  topLeft.h := theRect.left + titleBarHeight;
  topRight.v := topLeft.v;
  topRight.h := theRect.right - titleBarHeight;
  IF ((PtInRgn(topLeft,deskRgn)) OR
  (PtInRgn(topRight,deskRgn))) THEN
    OnAScreen := TRUE
  ELSE
    OnAScreen := FALSE;
END; (OnAScreen)

```

```

PROCEDURE ZoomIt(theWindow:WindowPtr;partCode:INTEGER;
                  clickedWhere:Point);
(ZoomIt supports more elegant window zooming on multiple
screen systems. theWindow will zoom to fill whatever
screen the zoom box was on when it was clicked. The
window state toggles between original size and zoomed
size, as usual. Thanks to Greg Marriott for this code.)

```

```

CONST
  titleBarHeight = 18;
TYPE
  WStatePtr = ^WStateData;
  WStateHandle = ^WStatePtr;
VAR
  oldRect,newRect: Rect;
  maxHeight: INTEGER;
BEGIN
  oldRect := theWindow^.portRect;
  IF theEnv.hasColorQD THEN
    BEGIN
      newRect := WhichDevice(clickedWhere)^^.gdRect;
      IF WhichDevice(clickedWhere) = GetMainDevice THEN
        newRect.top := newRect.top + MenuBarHeight;
    END
    newRect := GetGrayRgn^^.rgnBBox;
    newRect.left := newRect.left + 2;
    newRect.top := newRect.top + titleBarHeight + 2;
    newRect.right := newRect.right - 3;
    newRect.bottom := newRect.bottom - 3;
    IF NOT EqualRect(oldRect,newRect) THEN
      WITH WindowPeek(theWindow)^ DO
        WStateHandle(dataHandle)^^.stdState := newRect;
        SetPort(theWindow);
        EraseRect(whichWindow^.portRect);
        InvalRect(whichWindow^.portRect);
        ZoomWindow(theWindow,partcode,FALSE);
      END; (ZoomIt)

```

```
($I XCmdGlue.inc)
```

```
PROCEDURE Fail(errStr:Str255);
(Fail returns errStr to HyperCard and exits the XCMD.
errStr can then be checked by inspecting HyperCard's
global variable "the result." See "XCMD's for Hyper-
Card" by Gary Bond (MIS Press, 1988) for more details.
```

```
• 1988 by Gary Bond
All rights reserved.
You may use this code for NON-COMMERCIAL purposes.)
```

```
BEGIN
    paramPtr^.returnValue := PasToZero(errStr);
    SysBeep(1);
    EXIT(Window);
END; (Fail)
```

```
PROCEDURE CheckParamCount;
(CheckParamCount sees if the number of parameters
passed to the XCMD matches the number expected. If
not, we exit from the XCMD with an error message.
See "XCMD's for HyperCard" by Gary Bond (MIS Press,
1988) for more details.
```

```
• 1988 by Gary Bond
All rights reserved.
You may use this code for NON-COMMERCIAL purposes.)
```

```
VAR
    numParams: INTEGER;
BEGIN
    numParams := paramPtr^.paramCount;
    IF (numParams <> minParamCount) THEN
        Fail('Form: HyperWindow "Window
            Title",top,left,bottom,right');
END; (CheckParamCount)
```

```
FUNCTION GetHCVersion: Str255;
(Return a string containing the version of HyperCard
being used; e.g., '1.2')
```

```
BEGIN
    ZeroToPas(EvalExpr('the version')^,GetHCVersion);
END; (GetHCVersion)
```

```
PROCEDURE HideWindoids;
(Get and save the visible state of the tool, pattern,
message and fatbits windoids; then hide them if they
are showing.)
```

```
VAR
    toolH,path,msgH,fatH: Handle;
PROCEDURE HideFatBits;
(HyperCard does not have a built-in command for hiding
and showing the fatbits windoid, so we have to do it
ourselves. HideFatBits walks the window list until it
finds a window with title "FatBits," then hides it if
the visible field of its WindowRecord is true.
HideFatBits also saves the WindowPtr to the fatbits
windoid so we can use it later (e.g., to show the
windoid again.)
```

```
CONST
    windowList = $906; (Low memory global location.)
```

```
VAR
    theWindow: WindowPeek;
    theWindowPtr: ^WindowPtr;
```

```
BEGIN
    theWindowPtr := Pointer(windowList);
    theWindow := WindowPeek(theWindowPtr^);
    fatVis := FALSE;
    WHILE (theWindow <> NIL) DO
        BEGIN
            IF (theWindow^.titleHandle^ = 'FatBits') THEN
                BEGIN
                    fatBitsWindow := WindowPtr(theWindow);
                    IF (theWindow^.visible = TRUE) THEN
                        BEGIN
```

```
                            fatVis := TRUE;
                            HideWindow(fatBitsWindow);
                            theWindow := NIL;
                        END;
                    END;
                    IF (theWindow <> NIL) THEN
                        theWindow := WindowPeek(theWindow)^.nextWindow;
                    END; (WHILE)
                END; (HideFatBits)
```

```
BEGIN (HideWindoids)
```

```
    (Get visible state of windoids.)
```

```
    toolH := EvalExpr('visible of tool window');
    ZeroToPas(toolH^,toolStr);
    DisposHandle(toolH);
    toolVis := StrToBool(toolStr);
```

```
    path := EvalExpr('visible of pattern window');
    ZeroToPas(path^,patStr);
    DisposHandle(path);
    patVis := StrToBool(patStr);
```

```
    msgH := EvalExpr('visible of message window');
    ZeroToPas(msgH^,msgStr);
    DisposHandle(msgH);
    msgVis := StrToBool(msgStr);
```

```
    (Hide the ones that are showing.)
```

```
    HideFatBits;
```

```
    IF toolVis THEN
        SendHCMessage('hide tool window');
    IF patVis THEN
        SendHCMessage('hide pattern window');
    IF msgVis THEN
        SendHCMessage('hide message window');
END; (HideWindoids)
```

```
PROCEDURE ShowWindoids;
```

```
(This routine assumes HideWindoids has been called
before. ShowWindoids restores the visible state of
the windoids to that saved by HideWindoids.)
```

```
BEGIN
```

```
    IF toolVis THEN
        SendHCMessage('show tool window');
    IF patVis THEN
        SendHCMessage('show pattern window');
    IF msgVis THEN
        SendHCMessage('show message window');
```

```
(As in HideWindoids, we must take care of the FatBits
windoid ourselves.)
```

```
    IF fatVis THEN
        BEGIN
            ShowWindow(fatBitsWindow);
            SelectWindow(fatBitsWindow);
        END;
END; (ShowWindoids)
```

```
PROCEDURE ToggleMenuBar;
```

```
(Set the visible of the menubar to not the visible of
the menubar.)
```

```
BEGIN
    IF MenuBarHeight = 0 THEN
        SendHCMessage('show menubar')
    ELSE
        SendHCMessage('hide menubar');
END; (ToggleMenuBar)
```

```
PROCEDURE GetHCBitMap;
```

```

VAR
  myDocHeight, myDocWidth: INTEGER;
  HCRect, myHCRect: Rect;
  dummyGrafPtr: GrafPtr;
  HCBits: BitMap;
BEGIN
  MoveHHI(Handle(theOffScrHandle));
  HLock(Handle(theOffScrHandle));
  WITH theOffScrHandle^^ DO
    BEGIN
      myDocWidth := docWidth;
      myDocHeight := docHeight;
    END;
    myBits.rowBytes := (((myDocWidth - 1) DIV 16) + 1) * 2;
    myBits.baseAddr := NewPtr(myBits.rowBytes * myDocHeight);
    IF (myBits.baseAddr = NIL) THEN
      Fail('Could not allocate bitmap. ');
    HCRect := HCPort^.portRect;
    HCBits := HCPort^.portBits;
    SetRect(myBits.bounds, 0, 0, 528, 358);

    dummyGrafPtr := GrafPtr(NewPtr(sizeof(GrafPort)));
    IF (dummyGrafPtr = NIL) THEN
      BEGIN
        DisposPtr(myBits.baseAddr);
        Fail('Could not allocate offscreen grafPort. ');
      END;
      theOffScrHandle^^.gPort := dummyGrafPtr;
      OpenPort(theOffScrHandle^^.gPort);
      SetOrigin(0, 0);
      SetPortBits(myBits);
      MovePortTo(0, 0);
      PortSize(myDocWidth, myDocHeight);
      RectRgn(theOffScrHandle^^.gPort^.visRgn,
        theOffScrHandle^^.gPort^.portRect);
      RectRgn(theOffScrHandle^^.gPort^.clipRgn,
        theOffScrHandle^^.gPort^.portRect);
      EraseRect(theOffScrHandle^^.gPort^.portRect);
      SetRect(myHCRect, 0, 0, 512, 342);
      CopyBits(HCBits, theOffScrHandle^^.gPort^.portBits,
        HCRect, myHCRect, srcCopy, NIL);
      HUnlock(Handle(theOffScrHandle));
    END; (GetHCBitsMap)

PROCEDURE AdjustCursor;
  (AdjustCursor changes cursorRgn to the region that
  contains the cursor. As soon as the cursor moves out of
  cursorRgn, we get an event and can change the cursor and
  cursorRgn again. cursorRgn is either the content region
  of our window or the region containing everything BUT the
  content region of our window.)
VAR
  mousePt: Point;
  myWinContRect: Rect;
  myWinContRgn: RgnHandle;
  deskRgn: RgnHandle;
  handHdl: CursHandle;
BEGIN
  SetPort(myWindow);
  GetMouse(mousePt);
  LocalToGlobal(mousePt);
  myWinContRgn := NewRgn;

  (Calculate the "work region" of our window, which is its
  content region minus the scroll bars and grow icon.
  This is the region within which we want the cursor to
  change to HyperCard's browse tool, and outside of which
  we want it to be an arrow.)
  WITH WindowPeek(myWindow)^.contRgn^^.rgnBBBox DO
    SetRect(myWinContRect, left,
      top,
      right - 15,
      bottom - 15);
  RectRgn(myWinContRgn, myWinContRect);

```

```

IF PtInRect(mousePt, myWinContRect) THEN
BEGIN

```

```

  (The cursor is in the work region of our window. Set
  to browse tool.)

```

```

  handHdl := GetCursor(browseTool);
  IF (handHdl <> NIL) THEN
    SetCursor(handHdl^^)

```

```

ELSE

```

```

  (Set to arrow if can't find browse tool resource.)

```

```

  InitCursor;

```

```

  (Set the cursor region equal to our window's work
  region.)

```

```

  SetEmptyRgn(cursorRgn);
  CopyRgn(myWinContRgn, cursorRgn);

```

```

END

```

```

ELSE

```

```

BEGIN

```

```

  (The cursor is outside our window. Set to arrow.)

```

```

  InitCursor;

```

```

  (Get the current desktop region.)

```

```

  deskRgn := GetGrayRgn;

```

```

  (Set cursorRgn to the desktop region's bounding box.
  It is important to add the menu bar area to cursorRgn
  too.)

```

```

  SetRectRgn(cursorRgn, deskRgn^^.rgnBBBox.left,
    deskRgn^^.rgnBBBox.top,
    deskRgn^^.rgnBBBox.right,
    deskRgn^^.rgnBBBox.bottom);

```

```

  (Punch out our window's content region from the big
  region.)

```

```

  DiffRgn(cursorRgn, myWinContRgn, cursorRgn);

```

```

END;

```

```

  DisposeRgn(myWinContRgn);

```

```

END; (AdjustCursor)

```

```

BEGIN (Main Program)

```

```

  (Check the HyperCard version. Must be 1.2 or greater.)

```

```

  IF GetHCVersion < '1.2' THEN

```

```

    Fail('Sorry, must have HyperCard 1.2 or greater. ');

```

```

  (Save thy grafPort upon entering!)

```

```

  GetPort(HCPort);

```

```

  (Check and reset our environment.)

```

```

  CheckParamCount;
  FlushEvents(everyEvent, 0);
  InitCursor;

```

```

  (Find out what kind of machine we're running on.)

```

```

  envError := SysEnvirons(1, theEnv);

```

```

  IF (envError <> noErr) THEN

```

```

    Fail('SysEnvirons call failed. ');

```

```

  (Convert HyperTalk input parameters for use here.)

```

```

  ZeroToPas(paramPtr^.params[1]^, wTitle);

```

```

  ZeroToPas(paramPtr^.params[2]^, wT);

```

```

  ZeroToPas(paramPtr^.params[3]^, wL);

```

```

  ZeroToPas(paramPtr^.params[4]^, wB);

```

```

  ZeroToPas(paramPtr^.params[5]^, wR);

```

```

wTop := INTEGER(StrToNum(wT));
wLeft := INTEGER(StrToNum(wL));
wBottom := INTEGER(StrToNum(wB));
wRight := INTEGER(StrToNum(wR));

(If window size parameters are too small or illegal, set
 the window to a predefined minimum size.)

IF ((wRight - wLeft) < smallestWidth) THEN
    wRight := wLeft + smallestWidth;
IF ((wBottom - wTop) < smallestHeight) THEN
    wBottom := wTop + smallestHeight;

(Make sure the user is not trying to draw the window off
 the screen or under the menubar.)

SetRect(wRect, wLeft, wTop, wRight, wBottom);
IF NOT OnAScreen(wRect) THEN
    Fail('You are trying to draw your window off the
    screen!');

(Get the bounds of the desktop.)

screenRect := GetGrayRgn^.rgnBBox;

(If we have a small screen, make a note of it so we can
 hide the card window during context switches under
 MultiFinder.)

ZeroToPas(EvalExpr('item 3 of the screenRect'), widthStr);
screenWidth := INTEGER(StrToNum(widthStr));
ZeroToPas(EvalExpr('item 4 of the screenRect'),
    heightStr);
screenHeight := INTEGER(StrToNum(heightStr));

IF (screenWidth = 512) AND (screenHeight = 342) THEN
    smallScreen := TRUE
ELSE
    smallScreen := FALSE;

HideWindows;

IF MenuBarHeight > 0 THEN
    menuWasHidden := FALSE
ELSE
    menuWasHidden := TRUE;

theOffScrHandle := OffScrHandle(NewHandle
    (SizeOf(OffScrRecord)));
IF MemError <> noErr THEN
    Fail('Out of memory. Buy more.');
```

```

myDocWidth := HWidth + padding;
myDocHeight := HHeight + padding;

MoveHHi(Handle(theOffScrHandle));
HLock(Handle(theOffScrHandle));

WITH theOffScrHandle DO
BEGIN
    docWidth := myDocWidth;
    docHeight := myDocHeight;
END;

GetHCBITMap;

(Draw the window!)

myWindow := NewWindow(@wRecord, wRect,
    wTitle, TRUE, zoomDocProc,
    WindowPtr(-1), TRUE, 1);

IF (myWindow = NIL) THEN
BEGIN
```

```

ShowWindows;
Fail('Not enough memory to draw window.');
```

```

END
ELSE
BEGIN
    (This is for scrolling. Stay tuned...)
    SetWRefCon(myWindow, LONGINT(theOffScrHandle));
    InitBlt(myWindow);
    DrawGrowIcon(myWindow);
    theScrollHandle := ScrollHandle(NewHandle
        (SizeOf(ScrollRecord)));
    IF MemError <> noErr THEN
        Fail('Out of memory. Buy more.');
```

```

    (Draw horizontal and vertical scroll bars in our
    window.)

hScroll := CreateHScrollBar(myWindow, 0, 0,
    myDocWidth,
    LONGINT(theScrollHandle));
vScroll := CreateVScrollBar(myWindow, 0, 0,
    myDocHeight,
    LONGINT(theScrollHandle));

HiliteScrollBars(myWindow);
DrawContents(myWindow);
SetRect(dragRect, screenRect.left + 4,
    screenRect.top,
    screenRect.right - 4,
    screenRect.bottom - 4);
largestHeight := screenRect.bottom - screenRect.top;
largestWidth := screenRect.right - screenRect.left;
SetRect(winSizeLimits, smallestWidth,
    smallestHeight,
    largestWidth,
    largestHeight);

HRefresh := 'Go to this card';
cursorRgn := NewRgn;
inBackground := FALSE;
DoneFlag := FALSE;
REPEAT

    (Call WaitNextEvent, if available. Otherwise call
    GetNextEvent.)

    IF hasWaitNextEvent THEN
        HaveEvent := WaitNextEvent(everyEvent,
            myEvent, 30, cursorRgn)

    ELSE
    BEGIN
        HaveEvent := GetNextEvent(everyEvent, myEvent);
        AdjustCursor;
    END;
    IF HaveEvent THEN
    BEGIN
        IF (myEvent.what = app4Evt) THEN

            (Pre-process app4Evt's fed to us by MultiFinder.)

            CASE BSR(myEvent.message, 24) OF

                MouseMovedEvt:
                    AdjustCursor;

                SuspendResumeEvt:
                    BEGIN
                        myEvent.what := activateEvt;

                        (Resume event.)

                        IF (BAND(myEvent.message, SuspendEventMask) <>
                            0) THEN
                            inBackground := FALSE
                        ELSE
```

```

        (Suspend event.)

        inBackground := TRUE;

        myEvent.message := LONGINT(myWindow);
        END; {SuspendResumeEvt}

    END; {CASE BSR}
END; {IF HaveEvent}

CASE myEvent.what OF

mouseDown:
BEGIN
    partCode := FindWindow(myEvent.where, whichWindow);
    IF (whichWindow = myWindow) THEN
        BEGIN

            (Deal with mouse hits to our window.)

            CASE partCode OF

inDrag:
            BEGIN
                SelectWindow(whichWindow); (DragWindow bug)
                DragWindow(whichWindow, myEvent.where,
                    dragRect);
                SendCardMessage(HCrefresh);
                AdjustCursor;
            END;

inGrow:
            IF StillDown THEN (GrowWindow bug)
            BEGIN
                oldSize := whichWindow^.portRect;
                newSize := GrowWindow(whichWindow,
                    myEvent.where,
                    winSizeLimits);

                IF (newSize <> 0) THEN
                    BEGIN
                        InvalScroll(whichWindow);
                        SizeWindow(whichWindow,
                            LOWORD(newSize),
                            HIWORD(newSize), FALSE);
                        InvalContents(whichWindow, oldSize);
                        DrawGrowIcon(whichWindow);
                        MoveScrollbar(whichWindow);
                    END; {IF newSize}
                END; {IF StillDown}
            END; {inGrow}

inZoomIn, inZoomOut:
            BEGIN
                IF (TrackBox(whichWindow,
                    myEvent.where,
                    partCode)) THEN
                    BEGIN
                        InvalScroll(whichWindow);
                        ZoomIt(whichWindow, partCode,
                            myEvent.where);
                        InvalContents(whichWindow, oldSize);
                        DrawGrowIcon(whichWindow);
                        MoveScrollbar(whichWindow);
                    END; {IF TrackBox}
                END; {inZoomIn, inZoomOut}

inContent:
            BEGIN
                SetPort(whichWindow);
                eventPoint := myEvent.where;
                GlobalToLocal(eventPoint);
                controlCode := FindControl(eventPoint,
                    whichWindow,

```

```

                    whichControl);
                ClipRect(whichWindow^.portRect);
                IF (controlCode = inThumb) THEN
                    ScrollWithThumb(whichControl, eventPoint)
                ELSE IF (controlCode <> 0) THEN
                    dummy := TrackControl(whichControl,
                        eventPoint,
                        @MyScroll);
            END; {inContent}

inGoAway:
            BEGIN
                IF (TrackGoAway(whichWindow, myEvent.where))
                    THEN
                        DoneFlag := TRUE;
            END; {inGoAway}

        END; {CASE partCode}
    END; {IF whichWindow = myWindow}
END; {mouseDown}

activateEvt:
BEGIN
    IF (WindowPtr(myEvent.message) = myWindow) THEN
        BEGIN
            SetPort(myWindow);
            ClipRect(myWindow^.portRect);
            DrawGrowIcon(myWindow);
            IF inBackground THEN
                BEGIN
                    (We've been sent behind another application
                    under MultiFinder, so deactivate the scroll
                    bars and show the menubar if it was hidden.)
                    Deactivate(myWindow);
                    IF smallScreen THEN
                        ShowHide(WindowPtr(HCPort), FALSE);
                    IF MenuBarHeight = 0 THEN
                        BEGIN
                            SendHMessage('show menubar');
                            menuWasHidden := TRUE;
                        END
                    ELSE
                        menuWasHidden := FALSE;
                    END
                END
            ELSE
                BEGIN
                    (We've been brought to the front under Multi-
                    Finder, so reactivate the scroll bars and
                    hide the menubar if it was hidden before.)
                    ShowHide(WindowPtr(HCPort), TRUE);
                    IF menuWasHidden THEN
                        SendHMessage('hide menubar');
                        DrawControls(myWindow);
                        HiliteScrollbar(myWindow);
                        FlushEvents(everyEvent, 0);
                    END; {IF inBackground}
                END; {IF WindowPtr}
            END; {activateEvt}

updateEvt:
BEGIN
    (Handle updates to our window.)
    IF (WindowPtr(myEvent.message) = myWindow) THEN
        BEGIN
            (Always do this stuff)
            SetPort(myWindow);
            BeginUpdate(myWindow);
            ClipRect(myWindow^.portRect);
            DrawGrowIcon(myWindow);

            (Always do this stuff under single Finder but
            only if in foreground under MultiFinder)

            IF NOT inBackground THEN

```

```

BEGIN
  HiliteScrollBars(myWindow);
  DrawControls(myWindow);
  AdjustCursor;
END;

(Always do this stuff)

DrawContents(myWindow);
EndUpdate(myWindow);
END (IF myEvent.message)

(Handle updates to HyperCard's card window.)

ELSE IF (WindowPtr(myEvent.message) =
WindowPtr(HCPort)) THEN
BEGIN
  SendHCMMessage(HCRefresh);

  (Must zero out card window's update region
  ourselves because the SendHCMMessage call
  doesn't do it, and we'll wind up in an
  infinite loop if we don't.)

  BeginUpdate(WindowPtr(HCPort));
  EndUpdate(WindowPtr(HCPort));
END; (IF... ELSE)
END; (updateEvt)

keyDown:
BEGIN

```

```

IF (BitAnd(myEvent.modifiers,cmdKey) <> 0) THEN
BEGIN
  charCode := BitAnd(myEvent.message,
                     charCodeMask);
  (Pressing command-spacebar toggles the menubar.)
  IF (CHR(charCode) = ' ') THEN
    ToggleMenuBar;
    (Pressing command-W closes the window and exits
    the XCMD.)
    IF (CHR(charCode) = 'w') THEN
      DoneFlag := True;
    END;
  END;
END; (CASE myEvent.what)

UNTIL (DoneFlag = True);

(Clean up and get outta here!)
DisposeRgn(cursorRgn);
DisposeHandle(Handle(theOffScrHandle));
DisposeHandle(Handle(theScrollHandle));
CloseWindow(myWindow);
SendCardMessage(HCRefresh);
ShowWindows;
InitCursor;
FlushEvents(everyEvent,0);
(Restore thy grafPort)
SetPort(HCPort);
END; (IF myWindow...ELSE)
END; (Main)
END. (Window)

```



HyperChat™

XCMD Corner: HyperAppleTalk IV



HyperCard

Donald Koscheka
Arthur Young & Co.
MacTutor Contributing Editor
Volume 5, Number 4

I sometimes think that, to most programmers, the network is best left to the handful of specialists that ply their craft on this arcane art.

Think of the network as adding a new dimension to your programs, much as perspective enabled medieval artists to add depth and realism to their paintings. While many artists may have scoffed at perspective, I have no trouble believing that the populace was awed by its discovery. Perhaps networking will shine the same light as we enter the renaissance of computer programming.

If you've been following this column over the last three months, you may be asking yourself, "Are we there yet?" This month, we tie the pieces together by introducing the rest of AppleTalk XCMDs. I apologize for publishing listings last month that you can't really use until this month. I hope this issue makes up for it and in the future I will do my best to provide "soup to nuts" code in each issue.

Conceptually, an AppleTalk interface can be reduced to the six steps: (1) Open access to the transaction protocol and allocate any memory that will be needed to interface to it. I chose the AppleTalk Data Stream Protocol (ADSP). ADSPOpen hooks us up to ADSP, establishes a connection listener and returns the connection listener's socket address to Hypercard. NBPOpen initializes a data structure that we use to interface to NBP. The January '89 issue of this column provides the code for these XCMDs.

(2) Given the address of the connection listener, register a name and type with the network using the name binding protocol. Once registered, our entity's name and address will be listed in the "directory".

(3) If we want to be able to "dial up" other entities, we next perform a lookup which returns all current names in the lookup table which acts just like a directory. The names are returned as a list of items to Hypercard without their addresses. This step is optional, if all you expect to do is respond to requests (as in the case of a server), then you don't need to know the names of the requesting entities.

(4) To call another entity, invoke the ADSPCall XCMD with the entity's name. ADSPCall extracts the network address from the lookup table using the routine "NBPGetAddress". This technique allows us to de-couple names and addresses; your stacks need never be concerned with the address of an entity, the name is sufficient at the Hypercard script level.

(5) Once a connection is established with another entity (called the remote end), you send information to that entity using ADSPTalk. You receive information from the remote end using ADSPListen. ADSPListen is the heart of the interface, in addition to checking for incoming messages, it also monitors the

status of all open connections.

(6) When the session completes, remove your name from the names table using the NBPUnregister XFCN. Once unregistered, close down the interface to ADSP with the ADSPClose and then close down your NBP interface with NBPClose.

—The XCMDS—

While the process is conceptually quite simple, the amount of code needed to implement this interface may seem overwhelming. The rest of this article focuses on ADSPCall, ADSPTalk, ADSPListen and ADSPHangup. Information on both NBP and the open and close xcmts are available in that last two installments of this column as well as in recently published literature from APDA.

(1) ADSPCall

ADSPCall (listing 1) first extracts the address of the entity from the lookup table using NBPGetAddress. If you pass it the name of a currently visible entity, you have a reasonable assurance that the call will complete.

Upon retrieving the address, adspcall walks the connection list checking to see if a connection already exists with the remote end. If not, it creates a connection by calling DSPCreate.

Outgoing calls are placed asynchronously, control is passed back to Hypercard while the call completes in the "background". Asynchronous communications pays big dividends, adspcall attempts to establish the connection as many times as you request in the retry field. Waiting for a call to complete can "hang" the system, a particularly uncomfortable state for the user. If for any reason, the remote end has gone off line, Hypercard won't be stuck with an unbearable delay waiting for the call to timeout.

Outgoing calls, like all other transactions on the network, must be monitored by the listener. ADSPListen detects when the remote end picks up the phone.

To invoke ADSPCall from your script, use the following command line:

```
- get the name of the entity from the lookup table.  
- We let HyperAppleTalk handle the defaults for  
- retry, interval and buffer size.  
-  
- The two above globals will have been initialized by  
- NBPOpen and ADSPOpen respectively.  
-  
global globalNBPDData, globalADSPData  
ADSPCall theEntity
```

(2) ADSPTalk

Once a connection is established with a remote end, we can send messages to that entity. Like ADSPCall, ADSPTalk uses NBPGetAddress to get the address given a name. Because

NBPGetAddress returns 0 if the name had no corresponding address in the lookup table, we can determine whether we have a connection to the remote end. ADSPTalk scans the connection list to match the address with that of an entity in our connection list (pointed to by `adsp^.ends`). ADSPTalk calls DSPTalk (covered last month) to send the information. Outgoing messages are also sent asynchronously so that the user is not made to suffer any delays while the transmission completes.

Like ADSPCall, the method of invoking ADSPTalk is straightforward:

- "theEntity" is the name of the remote end that you
- called using ADSPCall and the card field contains the
- data that you wish to send to the remote end.

```
global globalNBPDData, globalADSPData
ADSPTalk theEntity, card field "outgoing message"
```

(3) ADSPListen

The quality of any conversation depends more on the skill of the listener than that of the talker. This is no less true for the network - talking is the easy part, the listener does the hard part. Like any good listener, ADSPListen must perform several tasks at once — (1) Answer incoming calls, (2) Monitor the status of outgoing calls, (3) Receive incoming messages and (4) periodically check the status of each connection.

Putting ADSPListen in the idle method of a card or stack provides a method for periodically checking up on each connection.

For each established connection an entry exists in our `adsp^.ends` list. While most of the data stored in the connection (cn) block is for our own use, the network keeps us apprised of the connection's status via two records: the parameter block (pb) and the connection control block (ccb). The parameter block behaves as it would for any low-level I/O on the Macintosh, it is the interface to the ADSP device driver. The connection block is used by adsp to keep us up to date on the status of each connection. In essence, we relinquish control of the parameter block and the connection control block to ADSP. The rest of the structures in the connection data are for our own use.

To determine what is going on in any connection, we check the state field in the connection control block. The following states are currently supported (see listing 2 for the corresponding code):

sClosed: The connection went down for some reason or other. At this point, we have no choice but to Hang up.

sOpen: A connection opens in stages. When we first create or detect an opening connection, we set the mode to NOP indicating the connection is only "half open". A half open connection is one in which only one of the two ends is ready. Receiving the sOpen state from the remote end informs us that our connection is now open.

A connection that is already open signals that it's closing with the "eClosed" message. If a connection is closing, hang up on it.

The next two tasks, talking and listening, are monitored by Checkmessages. Recall that ADSPTalk queues outgoing calls; they complete some time in the future. Similarly, incoming

messages can complete at any time. To determine whether any incoming or outgoing messages need attention, we issue a Status call to adsp. The `ioCrefNum` field of the `dspParamBlock` in `checkMessages` gets the driver number of the adsp driver. The `ccbRefnum` field gets the connection number of the connection we are checking.

The status call is issued synchronously, we must wait for it to complete before testing the status of a connection. If `recvQPending` is not 0, we have an incoming block of data. On input, we call `ReadADSPData` which will accumulate the data into the connection's input buffer until ADSP signals that there is no more data with the end-of-message flag (`eom`).

`ReadADSPData` will read data from the connection's input buffer into the handle we stored in `conn^.inbuf` (see listing 3). Accumulating data is a simple task: resize the handle by the number of bytes that we need to read in (given to us by `adsp` in `recvQPending`. If `recvQPending` is 0, there are no bytes to read). Once the handle is resized, point to the old end of the handle, and read the data directly into that are in memory. The read returns the exact number of bytes read in. If that number differs from `recvQPending` (under ordinary circumstances this won't happen), then resize the handle to reflect the actual number of bytes read in.

After reading the data, we check the end-of-message flag (`eom`) which ADSP sets only after it sends the last block of the message down stream. If `eom` is set, there is no more data to read. We can now send the data back to hypercard. If we know the name of the connection, we send it back to hypercard via the 'ADSPCaller' global container. This gives your stack a way of determining who sent the input data. The data itself is put in the hypercard container, 'HyperADSPData'. Once the data and name have been reported to Hypercard, we can optionally send a message back, perhaps to invoke some special input processing. If the connection stored a message in its 'callme' field, we now invoke that message via a callback.

Monitoring outgoing calls is a bit simpler. If the connection's `sendQPending` field is not 0, we still have outgoing data in the output buffer. In this case, don't do anything. If on the other hand, the send queue has emptied out and there was data in it (the connection `outbuf` field is not nil), then dispose the memory currently occupied by the output data.

The last task that ADSPListen must handle is responding to incoming calls. It does this by calling its `respondtoListen` Procedure. If a connection request is pending, the connection listener's `ioResult` will be set to `noErr` (a negative result means an error occurred, a positive result means that no connection requests are pending, the listener still has its ears to the wall).

If a connection request is pending, we can open up a connection to the remote end whose address is passed in the `remoteCID` field. Once the connection is established, we can try to discover its name by issuing a call to `NBPGetName` with the pending connection's name. Knowing the name of an incoming caller is optional so we don't take extraordinary efforts to discover the name. If however, you needed the entities name (as in the case of a game), then you can do a lookup if `NBPGetName` turns up empty-handed. This is an advanced feature and, in the

interest of space, I've omitted it. If you absolutely need this capability, drop me a line.

Once the pending request is serviced, the connection listener can put its ear back to the wall by requeuing the connection listener call. The information needed to queue a listening request is already stored in the connection listener's parameter block (adsp^pb). Requeuing the request becomes a simple call to PBCControl with the connections parameter block.

The following script, added to your card, will do the trick:

```
on idle
global globalNBPData, globalADSPData
global adspCaller - name of the caller global HyperADSPData
- where incoming data goes
-
ADSPListen "ShowData"
end idle

- Show data is the method to invoke to handle
- incoming data. Assuming that we have a card field
- named "show me", the following method will display the
- incoming message:
-
on ShowData
global HyperADSPData

put HyperAdspData into card field "show me"
end showdata
```

(4) ADSPHangup

We've already seen how to detect and respond to a connection hanging up on us. Use ADSPHangup (listing 4) to inform the remote end that the connection is going down. Once again, the drill is the same: get the address from the lookup table, check to see that a connection exists with this entity and call DSPHangup to inform the entity that the connection is going down. We don't remove the connection from the connection list at this point because we have to wait for the remote end to acknowledge our disconnection request. ADSPListen receives this acknowledgment in the form of an eClosed message. Once closed, we can tear down the connection and deallocate its memory (dspRemove).

```
- A simple script for hanging up a connection
-
on Disconnect
global HyperADSPData, globalNBPData

ADSPHangup theEntity
End Disconnect
```

You now have a reasonably complete interface to AppleTalk. ADSP and NBP both offer more features than I've been able to show you in a short time. Notably, the out of stream and forward reset features of ADSP enable some very powerful enhancements to the basic protocol. I chose not to discuss these features in this series because the average application can get by quite well without them. If, however, you are writing a network server in Hypercard, you may want to explore these features in more detail. I will be happy to present the code for a full implementation of ADSP. In keeping with the spirit of MacTu-

tor, contact me immediately to report any bugs in the code. I can be reached through this Magazine or on AppleLink (N0735) or MacNet.

Next month, I'm going to switch gears back into low and explore the file manager from within Hypercard.

Listing 1: ADSPCall.p

```
(*****
(*file: ADSPCall.p *)
(*
(* Establish a connection with *)
(* the entity whose name is *)
(* passed as a parameter. *)
(*
(* params[1] = Name of entity*)
(* params[2] = listen message *)
(* params[3] = number of retries*)
(* params[4] = retry interval *)
(* params[5] = Input buffer siz*)
(* params[6] = Output buffer siz*)
(*
(* If already connected, just *)
(* return with noErr *)
(*
(* Defaults: *)
(* Input Buffer = 578 *)
(* Output Buffer = 578 *)
(* Retry count = 0 *)
(* Retry interval = 0 *)
(* Out: the Result := error *)
(*message, empty if none. *)
(*
(* ----- *)
(* By: Donald Koscheka *)
(* Date: 9-Jan-89 *)
(* All Rights Reserved *)
(*
(*
(*****

*****
Build Sequence

pascal ADSPCall.p -o ADSPCall.p.o
link -m ENTRYPOINT -rt XCMD=1302 -sn Main=ADSPCall0
ADSPCall.p.o0
"(libraries)"Interface.o 0
-o YourStackNameHere

*****

($R-)
($S ADSPCall)
UNIT Koscheka;

(*****
INTERFACE
(*****

Uses MemTypes, QuickDraw, OSIntf,
ToolIntf, PackIntf, HyperXCmd,
AppleTalk, ADSP, adspxcmd;

Procedure EntryPoint( paramPtr : XCmdPtr );

(*****
IMPLEMENTATION
(*****
TYPE Str31 = String[31];

PROCEDURE ADSPCall( paramPtr: XCmdPtr ); FORWARD;
```

```

Procedure EntryPoint( paramPtr : XCmdPtr );
Begin
    ADSPCall( paramPtr );
End;

Procedure ADSPCall( paramPtr : XCmdPtr );
VAR
    adsp      : ADSPPtr;
    error, terr: OSErr;
    inBufSiz  : Integer;
    outBufSiz : Integer;
    retry     : Integer;
    interval  : Integer;
    eAddr     : AddrBlock;
    cn        : CBPtr;
    nbp       : NBPPtr;
    found     : BOOLEAN;

($I XCmdGlue.inc )
($I XCMDADSP.inc )

(*****)
BEGIN
    error      := noErr;
    found      := TRUE; (** true if already established **)

WITH paramPtr^ DO
BEGIN
    IF (paramCount <> 0) AND (params[1]^ <> NIL) THEN
        BEGIN
            adsp := ADSPPtr(RetrieveData( 'GLOBALDSPDATA' ));
            IF adsp <> NIL THEN
                BEGIN
                    adsp^.checkPoint := RECEIVING;

                    (**** Before we go too far, let's****)
                    (*** look at some of the settings ****)
                    (*** set the retransmit count ****)
                    IF params[3] <> NIL THEN
                        retry := StringToShort( params[3]^ )
                    ELSE
                        retry := 0;

                    (*** Set the retransmit interval ****)
                    IF params[4] <> NIL THEN
                        interval := StringToShort( params[4]^ )
                    ELSE
                        interval := 0;

                    (*** Set the input buffer size ****)
                    IF params[5] <> NIL THEN
                        inBufSiz := StringToShort( params[5]^ )
                    ELSE
                        inBufSiz := ATPBSIZE;

                    (*** Set the output buffer size ****)
                    IF params[6] <> NIL THEN
                        outBufSiz := StringToShort( params[6]^ )
                    ELSE
                        outBufSiz := ATPBSIZE;

                    (*** (1) Get the entity's address****)
                    HLock( Handle( params[1]) );
                    eAddr := NBPGetAddress( params[1]^ );
                    HUnlock( Handle( params[1]) );

                    (*** Walk through the connection ****)
                    (*** list to see if the connection ****)
                    (*** already exists. ****)

                    IF LongInt(eAddr) <> 0 THEN
                        BEGIN
                            cn := adsp^.ends;
                            found := FALSE;

```

```

WHILE (NOT found) DO
    IF cn = NIL THEN
        found := TRUE
    ELSE
        IF LongInt( eAddr ) = LongInt( cn^.adr ) THEN
            found := TRUE
        ELSE
            cn := cn^.next;

    (*** If connection already exists****)
    (*** cn will not be NIL so we won't ****)
    (*** have to reconnect the two ends,****)
    (*** just fall out, otherwise, try to****)
    (*** make the connection. ****)

    IF cn = NIL THEN (*** establish connection ***)
        BEGIN
            cn^.remName := NBPGetName( eAddr );
            cn := DSPCreate(eAddr, inBufSiz,
                            outBufSiz, retry, interval, params[2] );
            IF cn <> NIL THEN
                BEGIN
                    cn^.remName := NBPGetName( eAddr );

                    (*** try to discover the name****)
                    IF cn^.remName = NIL THEN
                        BEGIN
                            nbp := NBPPtr(RetrieveData('GLOBALNBPDATA'));
                            IF nbp <> NIL THEN
                                BEGIN
                                    SendCardMessage('DoALookUp');
                                    cn^.remName := NBPGetName( eAddr );
                                END;
                            END;

                        (*** Now "dial for dollars" ****)
                        WITH cn^.dspPB DO
                            BEGIN
                                ioCRefNum := adsp^.dspRefNum;
                                ccbRefNum := cn^.ccbRef;
                                csCode := dspOpen;
                                ocMode := ocRequest;
                                ioCompletion := NIL;
                                remoteCID := 0; (** don't know this yet**)
                                localCID := 0; (** adsp allocates id**)
                                remoteAddress := eAddr;
                                filterAddress.aNet := 0;
                                filterAddress.aNode := 0;
                                filterAddress.aSocket := 0;
                                SendSeq := 0;
                                RecvSeq := 0;
                                SendWindow := 0;
                                attnSendSeq := 0;
                                attnRecvSeq := 0;
                                ocInterval := INTERVAL;
                                ocMaximum := RETRY;
                                error := PBControl( @cn^.dspPB, ASYNC );
                            END;
                        END;
                    END;
                    (LongInt(eAddr) <> 0 )
                    adsp^.checkPoint := CLOSE_OK;
                END;
            END;
            returnValue := PasToZero( NumToStr( LongInt(error) ) );
        END; (** with paramPtr^ **)
    END;

end.

```

Listing 2: ADSPTalk.p

```

(*****)

```

```

(*file:  ADSPTalk.p  *)
(*
(* Send information to an *)
(* established connection. *)
(* The connection must be *)
(* been established via the *)
(* ADSPCALL xcmd. *)
(*
(* In: *)
(* params[1] = entity name*)
(* of the remote end. *)
(*
(* _____ *)
(* By:  Donald Koscheka *)
(* Date: 2-Mar-89 *)
(* All Rights Reserved *)
(*
(* _____ *)
(*****

(*****
Build Sequence

pascal -o ADSPTalk.p.o ADSPTalk.p
link -m ENTRYPOINT -rt XCMD=1305 -sn Main=ADSPTalk
ADSPTalk.p.o
  "(libraries)"Interface.o o
  -o YourStackNameHere

*****

($R-)
($S ADSPTalk)
UNIT DummyUnit;

(*****
INTERFACE
(*****

Uses MemTypes, QuickDraw, OSIntf,
ToolIntf, PackIntf, HyperXCmd,
AppleTalk, ADSP, adspxcmd;

Procedure EntryPoint( paramPtr : XCmdPtr );

(*****
IMPLEMENTATION
(*****
TYPE Str31 = String[31];

PROCEDURE ADSPTalk( paramPtr : XCmdPtr ); FORWARD;

Procedure EntryPoint( paramPtr : XCmdPtr );
Begin
  ADSPTalk( paramPtr );
End;

Procedure ADSPTalk( paramPtr : XCmdPtr );
VAR
  adsp : ADSPPtr;
  error : OSErr;
  eAddr : AddrBlock;
  cb : cbPtr;

($I XCmdGlue.inc )
($I XCMDADSP.inc )

BEGIN
  error := noErr;
  adsp := ADSPPtr(RetrieveData( 'GLOBALDSPDATA' ));
  IF(adsp <> NIL)
  AND (paramPtr^.params[1] <> NIL)
  AND (paramPtr^.params[2] <> NIL) THEN

```

```

BEGIN
  adsp^.checkPoint := RECEIVING;
  (** (1) Get the entity's address **)
  HLock( paramPtr^.params[1] );

  eAddr := NBPGetAddress( paramPtr^.params[1] );
  cb := adsp^.ends;

  (** (2) Find entity in connection list**)

  IF LongInt( eAddr ) <> 0 THEN
    WHILE cb <> NIL DO
      IF (LongInt(cb^.adr)=LongInt(eAddr))
      AND (cb^.ccb.state=sOpen) THEN
        BEGIN
          error := DSPTalk(cb, paramPtr^.params[2] );
          cb := NIL;
        END
      ELSE
        cb := cb^.next;

    adsp^.checkPoint := CLOSE_OK;
    HUnlock( Handle(paramPtr^.params[1]) );
  END;
  paramPtr^.returnValue := PasToZero( NumToStr( LongInt(error)
  );
END;

end.

```

Listing 3: ADSPListen.p

```

(*****
(*file: ADSPListen.p  *)
(*
(* Perform the following: *)
(* (1) Listen for incoming *)
(* calls on the connection *)
(* listener *)
(* (2) Check to see if any *)
(* outgoing calls have been*)
(* answered *)
(* (3) Check connections *)
(* for incoming data *)
(* (4) Check connections for*)
(* attention messages *)
(*
(* In: *)
(* params[1] = name of *)
(* message to call hypercard*)
(* with on incoming data *)
(*
(* Defaults: *)
(* container = HyperADSPData*)
(* message = none. *)
(*
(* _____ *)
(*All Rights Reserved *)
(*
(* _____ *)
(* By:  Donald Koscheka *)
(* Date: 2-Mar-89 *)
(* All Rights Reserved *)
(*
(* _____ *)
(*****

(*****
Build Sequence

pascal -o "(hpo)"ADSPListen.p.o "(adsp)"ADSPListen.p
link -m ENTRYPOINT -rt XCMD=1303 -sn Main=ADSPListen
ADSPListen.p.o
  "(libraries)"Interface.o o

```

```

-o YourStackNameHere

*****

($R-)
($S ADSPListen)
UNIT Koscheka;

(*****
    INTERFACE
(*****

Uses MemTypes, QuickDraw, OSIntf, ToolIntf, PackIntf, HyperXCmd, AppleTalk, ADSP, adspxcmd;

Procedure EntryPoint( paramPtr : XCmdPtr );

(*****
    IMPLEMENTATION
(*****
TYPE Str31 = String[31];

PROCEDURE ADSPListen( paramPtr: XCmdPtr ); FORWARD;

Procedure EntryPoint( paramPtr : XCmdPtr );
Begin
    ADSPListen( paramPtr );
End;

Procedure ADSPListen( paramPtr : XCmdPtr );
VAR
    myFlags : SignedByte;
(* flags from incoming unsolicited message *)
    adsp : ADSPPtr; (* pointer to ADSP Control block *)
    error : OSErr;
    err : OSErr;
    cn,nn : cbPtr; (* pointers to established connections *)
    nbp : NBPPtr; (* pointer to the memory NBP interface *)
    mybPtr : Ptr; (* used for getting and setting flags *)

($I XCmdGlue.inc )
($I XCMDADSP.inc )

PROCEDURE RespondToListen;
(*****
(* If the connection listener *)
(* has completed, then someone *)
(* is trying to call. If we can*)
(* initialize a connection, *)
(* acknowledge receipt of call *)
(*****
VAR
    err : Integer;
    conn : CBPtr;
    th : Handle;

BEGIN
    IF adsp^.pb.ioResult = noErr THEN
        BEGIN (** We Have an incoming call.**)
            (** Create a connection end to**)
            (** the caller and set the **)
            (** state to active indicating **)
            (** the connection is accepted**)
            conn := DSPCreate(adsp^.pb.remoteAddress,
                ATPBSIZE,ATPBSIZE,RETRY,INTERVAL,NIL );

            IF conn <> NIL THEN
                WITH conn^.dspPB DO
                    BEGIN (** accept the connection request**)
                        ioCRefNum := adsp^.dspRefNum;
                        ioCompletion := NIL;
                        csCode := dspOpen;
                        ocMode := ocAccept;
                        ccbRefNum := conn^.ccbRef;

```

```

remoteCID := adsp^.pb.remoteCID;
localCID := 0;
remoteAddress := adsp^.pb.remoteAddress;
filterAddress.aNet := 0;
filterAddress.aNode := 0;
filterAddress.aSocket := 0;
RecvSeq := 0;
attnRecvSeq := 0;
SendSeq := adsp^.pb.SendSeq;
attnSendSeq := adsp^.pb.attnSendSeq;
SendWindow := adsp^.pb.SendWindow;
ocInterval := INTERVAL;
ocMaximum := RETRY;
err := PBControl( @conn^.dspPB, ASYNC );

(** Call back Hypercard to inform **)
(** it of an incoming call. **)

conn^.remName := NBPGetName( adsp^.pb.remoteAddress
);
END;

(** Reset connection listener to**)
(** wait for more calls **)
err := PBControl( @adsp^.pb, ASYNC );
END;

PROCEDURE ReadADSPData( conn:CBPtr; cnt:Integer; callMe:Handle
);
(*****
(* Poll the connection to *)
(* determine whether any *)
(* data is pending. If so, *)
(* read it and pass it back to *)
(* hypercard via the callback*)
(* mechanism. *)
(* *)
(* IN: *)
(* conn : pointer to connection *)
(* cnt : num. of bytes to read *)
(* callMe : callback method *)
(*****
VAR
    Tempb : dspParamBlock;
    err : Integer;
    temp : LongInt;
    inSize : LongInt;
    globName: Str255;
    tp : Ptr;
    th : Handle;

BEGIN
    globName:= 'HyperADSPData';

    WITH conn^ DO
        BEGIN (** read the data **)
            IF inBuf = NIL THEN
                BEGIN
                    inBuf := NewHandle( LongInt(cnt) );
                    MoveHHI( inBuf );
                    HLock( inBuf );
                    tp:= inBuf^;
                END
            ELSE (** inBuf already allocated, add to end of it**)
                BEGIN
                    HUnlock( inBuf );
                    inSize := GetHandleSize(inBuf);
                    temp:= inSize;
                    inSize := inSize + LongInt( cnt );
                    SetHandleSize( inBuf, inSize );
                    MoveHHI( inBuf );
                    HLock( inBuf );
                    temp:= temp + Ord4( inBuf^ );

```

```

    tp := Pointer(temp);
END;

WITH Tempb DO (** now read the data **)
BEGIN
    ioCRefNum := adsp^.dspRefNum;
    ccbRefNum := conn^.ccbRef;
    csCode := dspRead;
    reqCount := cnt;
    dataPtr := tp;
END;
err := PBControl( @Tempb, SYNC );
HUnlock( inBuf );

IF err=noErr THEN
BEGIN
    inSize := GetHandleSize( inBuf );
    inSize := inSize - (cnt-Tempb.actCount);
    SetHandleSize( inBuf, inSize );
END;

IF (err=noErr) AND (Tempb.eom=1) THEN
BEGIN
    IF inBuf <> NIL THEN (**Call back with data**)
    BEGIN
        IF remName <> NIL THEN
            SetGlobal( 'ADSPCaller', remName );

            tp:= Pointer( Ord4( inBuf^ ) + LongInt(inSize) );
            tp^ := 0;

            SetGlobal( globName, inbuf );
            DisposHandle( inbuf );
            inbuf := NIL;
        END;

        IF callMe <> NIL THEN (**Set remote-end name**)
        BEGIN
            HLock( callMe );
            ZeroToPas( Pointer( callMe^ ), globName );
            HUnlock( callMe );
            SendCardMessage( globName );
        END;
    END;
END;(** read the data **)
END;

Function CheckMessages( cb : cbPtr ): OSErr;
(***)
(* check this connection for: *)
(* (1) Completed Incoming Messages *)
(* (2) Completed Outgoing Messages *)
(***)
VAR
    myErr : OSErr;
    tpb : dspParamBlock;
BEGIN
    WITH tpb DO
    BEGIN
        csCode := dspStatus;
        ioCRefNum := adsp^.dspRefNum;
        ccbRefNum := cn^.ccbRef;
    END;
    myErr := PBControl( @tpb, SYNC );

    IF myErr = noErr THEN
    BEGIN
        (** (3) Check for pending incoming data**)
        IF tpb.recvQPending <> 0 THEN
            ReadADSPData( cn, tpb.recvQPending, paramPtr^.params[1]
);

        (**(4) See if any write operations have completed**)
        IF (tpb.SendQPending = 0) AND (cn^.OutBuf <> NIL ) THEN

```

```

        BEGIN
            HUnlock( cn^.outBuf );
            DisposHandle( cn^.outBuf );
            cn^.outBuf := NIL;
        END;
    END; ( myErr <> noErr )
    CheckMessages := myErr;
END;

(***)
(* ADSPListen *)
(***)

BEGIN
    error := noErr;
    adsp := ADSPPtr(RetrieveData( 'GLOBALDSPDATA' ));

    IF adsp <> NIL THEN
    BEGIN
        adsp^.checkPoint := RECEIVING;
        cn := adsp^.ends;

        (**(2) Monitor status on all connections**)
        WHILE cn <> NIL DO WITH cn DO
        BEGIN
            nn := next;
            err := noErr;

            CASE ccb.state OF ( connection states )
                sClosed: err := DSPHangUp( cn );

                sOpen:
                BEGIN
                    IF mode = NOP THEN (**connection is opening **)
                    BEGIN (**inform Hypercard of connection**)
                        SetGlobal( 'ADSPCaller', remName );
                        SendCardMessage( 'ConnectionMade' );
                        mode := EST;
                    END
                    ELSE
                    BEGIN (** Check an established connection **)
                        myBPtr := @cn^.ccb.userFlags;

                        IF BAND(myBPtr^, eClosed)=eClosed THEN
                            err := DSPHangUp( cn )
                        ELSE
                        BEGIN
                            err := CheckMessages( cn );

                            myBPtr^ := 0; ( clear user flags )
                        END;
                    END;
                END; ( case of connection states )

                IF err <> noErr THEN error := err;
                cn := nn;
            END;
            RespondToListen; (** check for opening calls **)
            adsp^.checkPoint := CLOSE_OK;
        END; (** IF adsp <> NIL **)

        paramPtr^.returnValue := PasToZero( NumToStr( LongInt(err) )
);
    END;
end.

```

Listing 4: ADSPHangUp.p

```

(***)
(*file: ADSPHangUp.p *)
(* *)
(* Disconnect an established*)

```

```

(* connection and deallocate*)
(* the data associated with*)
(* the connection. *)
(* *)
(* In: params[1] Name of *)
(* the entity to hangup on *)
(* *)
(* Out: error message, *)
(* empty if none. *)
(* *)
(* *)
(* By: Donald Koscheka *)
(* Date: 2-Mar-89 *)
(* All Rights Reserved *)
(* *)
(* *)
(*****
Build Sequence

pascal -o ADSPHangUp.p.o ADSPHangUp.p
link -m ENTRYPOINT -rt XCMD=1304 -sn Main=ADSPHangUp0
ADSPHangUp.p.o0
"(libraries)"Interface.o 0
-o YourStackNameHere

*****

($R-)
($S ADSPHangUp)
UNIT DummyUnit;

(*****
INTERFACE
(*****

Uses MemTypes, QuickDraw, OSIntf,
ToolIntf, PackIntf, HyperXCmd,
AppleTalk, ADSP, adspxcmd;

Procedure EntryPoint( paramPtr : XCmdPtr );

(*****
IMPLEMENTATION
(*****
TYPE Str31 = String[31];

PROCEDURE ADSPHangUp( paramPtr: XCmdPtr ); FORWARD;
Procedure EntryPoint( paramPtr : XCmdPtr );

```

```

Begin
ADSPHangUp( paramPtr );
End;

Procedure ADSPHangUp( paramPtr : XCmdPtr );
VAR
adsp : ADSPPtr;
error : OSErr;
pb : DSPParamBlock;
eAddr : AddrBlock;
cb : CBPtr;

($I XCmdGlue.inc )
($I XCMDADSP.inc )
(*****
BEGIN
error := noErr;
IF ( paramPtr^.paramCount > 0 )
AND ( paramPtr^.params[1] <> NIL ) THEN
BEGIN
adsp := ADSPPtr(RetrieveData( 'GLOBALDSPDATA' ));
IF adsp <> NIL THEN
BEGIN
adsp^.checkPoint := RECEIVING;
(** (1) Get the entity's address**)
HLock( Handle(paramPtr^.params[1]) );
eAddr := NBPPGetAddress( paramPtr^.params[1]^ );
cb := adsp^.ends;

(** match the entity to the list **)
IF LongInt(eAddr) <> 0 THEN
WHILE cb <> NIL DO
IF LongInt(cb^.adr) = LongInt( eAddr ) THEN
BEGIN
error := DSPHangUp( cb );
cb := NIL;
END
ELSE
cb := cb^.next;

HUnlock( Handle(paramPtr^.params[1]) );
adsp^.checkPoint := CLOSE_OK;
END
END;
paramPtr^.returnValue := PasToZero( NumToStr( LongInt(error)
) );
END;

end.

```



HyperChat™

Menu Bar XCMD



Fred Stauder
HyperChat Editor
Zurich, Switzerland
Volume 5, Number 5

on HyperChat

Your wish is my XCMD

I have two XCMD's this month that I find quite useful. The first I called invisMenuBar, it makes the menubar invisible and still functional. It is really useful when developing on a small screen. The Human Interface Guidelines clearly state "don't hide things from the user and don't surprise them". So why am I showing you this XCMD? Firstly I would strongly object to people using it in a stack. It is a development/power user tool it may help you develop stacks and browse through other peoples if you have a small screen. Just like MacsBug deviates a long way from the guidelines but is highly useable and appropriate in it's context. What it does. It simply uses a callback to hypertalk to hide the menubar, then it sets the height of the menubar to the height of it's natural state. Hypercard is fooled to think that it is hidden so it doesn't do a menubar redraw, however when you have a mousedown in the menubar it accepts it and draws the menu.

(*
invisMenuBar- a sHyperCard external command
that makes the menubar invisible yet functional.

Written in MPW Pascal.
Written by Fred Stauder
©All rights reserved 1988

pascal :Sources:XCMD:InvisMenuBar.p
link -o AJS-60:test -rt XCMD=7004 -sn d
Main=InvisMenuBar :Sources:XCMD:InvisMenuBar.p.o d
o -m ENTRYPOINT

*)

InvisMenuBar
Segment name must be the same as the command name.

UNIT Fred_Stauder;

INTERFACE

USES MemTypes, QuickDraw, OSIntf, ToolIntf, PackIntf,
PasLibIntf, HyperXCMD;

PROCEDURE ENTRYPOINT(paramPtr:XCMDPtr);

IMPLEMENTATION

TYPE Str31 = String[31];

PROCEDURE InvisMenuBar(paramPtr:XCMDPtr); FORWARD;

PROCEDURE ENTRYPOINT(paramPtr:XCMDPtr);

BEGIN

InvisMenuBar(paramPtr);

END;

PROCEDURE InvisMenuBar(paramPtr:XCMDPtr);

```
XCMDGlue.inc the glue routines
VAR menuBarHeight: Ptr;

BEGIN
    SendCardMessage('hide menubar');
    menuBarHeight := Ptr($BAB);
    directly to the global the only way possible
    menuBarHeight* := $14;
    END;

END.
```

Listing 1.

The second XCMD allows you to change the window type. This is useful when you want to do a presentation and not have the window title visible. Or if you want to make a small window for a specific purpose. (fig.1.) The windowDefproc and the high order byte holds the window variation code. First you pass a parameter with the XCMD changeWindowType, that gives you the desired window. the types are the same as in Inside Mac V I p273.

documentProc = 0 (standard document window)
dBoxProc = 1 (alert box or modal dialog box)
plainDBox = 2 (plain box)
altDBoxProc = 3 (plain box with shadow)
noGrowDocProc = 4 (document window without size)
rDocProc = 16(rounded-corner window)

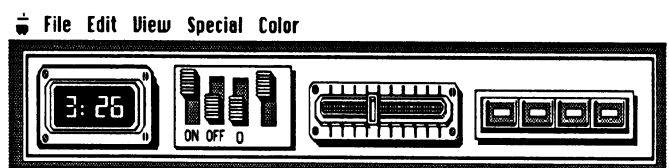


Figure 1.

You take the parameter, convert to a Longint, then comes the slightly tricky bit where you have to do bit manipulation to write it to the high order byte of the windowDefproc. You can get the window type by using the function GetWVariant. I will leave this as an exercise for you to write, so that once you have changed the window type you will be able to find out what it is through an XFCN.

Diagram 1. shows how the bits are manipulated to put the window variant into high order byte of theWindow^.DefProc handle. Then I hide and show the cardwindow to update the screen.

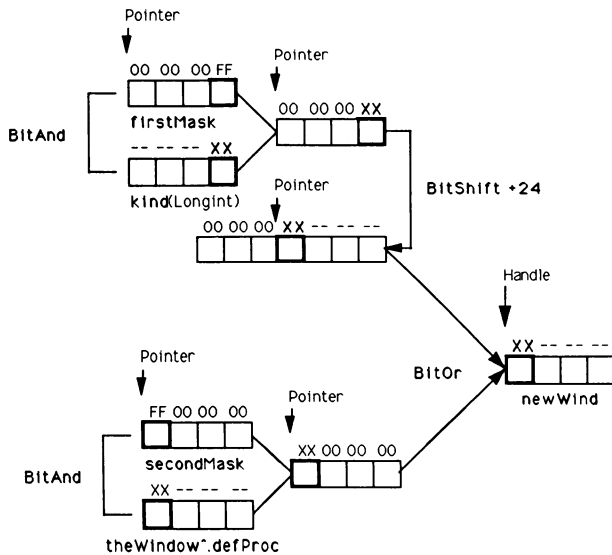


Diagram 1.

Listing 2.

Written in MPW Pascal.
Written by Fred Stauder
©All rights reserved 1988

```
(*
pascal :changeWindowType.p
link -o Development:Demo:home -rt XCMD=7005sn a
Main=changeWindowType :changeWindowType.p.o -m ENTRYPOINT
*)
```

changeWindowType name must be the same as the command name

```
UNIT DummyUnit;
INTERFACE
```

```
USES MemTypes, QuickDraw, OSIntf, ToolIntf, PasLibIntf,
HyperXCmd;
```

```
PROCEDURE ENTRYPOINT(paramPtr:XCmdPtr);
```

```
IMPLEMENTATION
```

```
Str31 = String[31];
```

```
PROCEDURE changeWindowType(paramPtr:XCmdPtr);
FORWARD;
```

```
PROCEDURE ENTRYPOINT(paramPtr:XCmdPtr);
BEGIN
changeWindowType(paramPtr);
END;
```

```
PROCEDURE changeWindowType(paramPtr:XCmdPtr);
VAR
HCWindow: WindowPtr;
theWindow: WindowPeek;
tempStr: Str255;
kind: Longint;
newWind: Longint;
firstMask: Longint;
firstShift: Longint;
secondMask: Longint;
```

XCmdGlue.inc the glue routines

```
BEGIN
GetPort(HCWindow);
thy port
ZeroToPas(paramPtr^.params[11], tempStr);
kind := StrToNum(tempStr);

theWindow := WindowPeek(HCWindow);
firstMask := BitAnd(Kind, $000000FF);
firstShift := BitShift(firstMask, 24);
secondMask :=
BitAnd(LONGINT(theWindow^.windowDefProc), $0FFFFFFF);
```

```
newWind := BitOr(firstShift, secondMask);
```

```
theWindow^.windowDefProc := Handle(newWind);
SendHMessage('hide cd window');
SendHMessage('show cd window');
SetPort(HCWindow);
thy port
```

```
END;
```

```
END.
```

Next month I will show you an XCMD that will allow you to resize the card window of Hypercard such as in Fig. 1.

Send articles ideas, comments etc to Applelink: STAUDER.

end HyperChat



HyperChat™

XCMD Corner: XCMDs in Think C



HyperCard

Donald Koscheka
Arthur Young & Company
MacTutor Contributing Editor
Volume 5, Number 5

Exploring XCMDs using just one development system is a lot like learning to play music on just one instrument. This is unfortunate because the Macintosh offers a virtual orchestra of development systems. As in learning to play a second instrument, learning a new development system often boils down to nothing more than learning how to transcribe the music.

Your programming environment is your instrument (writing a program is more like writing a piece of music than playing it). If you happen to own a copy of MPW you have no trouble using the code in this column because it's all written for MPW. If you use a different development system, you'll need to transcribe my listings to play on your particular development system.

In the past, I've been terribly guilty of catering to the MPW clique. My reasons were perhaps not so subtle: I use MPW at work; I'm comfortable with it, and I know that it directly supports an interface to XCMDs. After purchasing a copy of MPW 3.0, I realized that I might be making a mistake. For starters, MPW requires an expensive and time consuming learning curve; perhaps not everyone wants to spend several hundred dollars for sophistication they may never need.

Other development systems offer features that are not available in MPW such as a high quality source level debugger. Unfortunately several development systems lack key capabilities like support for XCMDs. Although you might expect LightspeedC to support XCMDs out of the box, it doesn't. I'm not sure I understand the reason, but it doesn't matter. Adding XCMD support to any compiler should be a very simple job.

Afterall, an XCMD is just another type of resource. If LightspeedC can create specialized resources such as window definitions and drivers, it already contains some of the support we need.

• Setting up the project

As luck, or design foresight, would have it, LightspeedC supports a generalized resource type called the "CODE" Resource. CODE resource projects are created like any other project in LightspeedC with the exception that you specify the project as a code resource in the set project type dialogue (see figure 1). You specify most of the information that the resource manager wants in this dialogue.

The custom header option requires a little elaboration. The standard interface to code resources branches to the entry point: `main()`. Selecting the Custom header option causes the entry point to be the first function in the file in which your `main()` is defined. XCMDs follow the second course as a matter of style.

Set the resource type to 'XCMD' or 'XFCN' according to

Figure 1. Setting up a Code Resource using the Set Project Type Dialogue

the type of Hypercard interface you want.

• Writing the XCMD

In LightspeedC, a code resource has the same form as a "C" program. You define the entry point as:

```
pascal void main( paramPtr )
```

ParamPtr is a pointer to a HyperTalk XCMDBlock. From there, your code looks like a vanilla "C" application. The same restrictions apply as on all CODE resources: no globals or statically initialized strings. CODE resources, of any kind, have no knowledge of globals at build time because they can't make any assumptions about which application will load them! Strings suffer the same fate - "C" allocates statically defined strings in the application's global pool. You're not building an application so you don't have a global pool - you must either define strings the hard way or load them in from the resource fork.

Listing 1 is a simple XCMD that returns the string "Hello World" to Hypercard. While the code itself is wholly unimaginative, it does demonstrate that the interface to Hypercard and the callback mechanism work satisfactorily. Anyhow there's a time for imagination and a time to just get things done. Porting an interface falls to the latter category.

Include the header file, "HyperXCMD.h" (listing 2) in your code. This is the standard interface to Hypercard transposed for LightspeedC. There aren't a lot of differences between the two as should be the case. Nonetheless this exercise proves quite useful in scoping out the idioms of a particular language implementation.

• The Hypercard "Glue"

You need to create a project which at the least includes MacTraps, your XCMD code and a special file called "XCMDGlue.c" (Listing 3) which interfaces (or glues) your XCMD to Hypertalk's callback mechanism. I took the liberty of translating XCMDGlue.in.c from MPW "C" to LightspeedC. The major difference between the MPW and Think versions of the glue is that I use the CallPascal function available in LightspeedC to jump to the subroutine pointed to by paramPtr->entryPoint. Pascal routines push parameters from left to right and the subroutine is responsible for clearing the stack parameters. "C" pushes from right to left and the caller clears the stack.

We know that the callback engine uses the Pascal calling sequence because the parameters aren't left on the stack after the call. Whether we push from right to left or left to right isn't relevant here for an obvious reason that's left as an exercise to the reader.

Add XCMDGlue.c to the project rather than including it at the end of the XCMD source code as is the case with MPW. I like this feature of Think "C" - you keep track of the source modules at the project level and not at the source code level.

• Creating the resource

Once all modules compile, use the "Create Code Resource" menu option to create the code resource. LightspeedC presents a dialogue asking for the name of an output file. Enter the name of a file but not your output stack: LightspeedC completely erases the previous contents of both forks of the output file before writing out the code resource!

Select the "smart link" option when creating a code resource. Your links will be slower, but your XCMDs will be quite a bit smaller (The example in listing 3 compiles to 12.5K with smart link of and 2.5K with smart link on). Of course, you can speed up turnaround during development by leaving this option unselected.

That's it! Use ResEdit or Rescopy to copy the XCMD into your stack. Perhaps LightspeedC will release a future version of "C" that will build code resources directly into the target file (if you try this now, the entire contents of the target file will be lost). In the meantime, the extra step needed to copy the resource into your stack is a small price to pay for an outstanding development system.

I hope the information in this article will help those of you who need to create an XCMD interface for another development system. The process is really rather simple and it provides you one of those rare opportunities in programming where you can get a lot done without doing a lot of work!

Listing 1:

```
/* File: SimpleXCMD.c */
/*
 * This is what a simple XCMD*
 * written in Lightspeed "C" *
 * In order to build this code *
 * resource, you will need the *
 * two files "HyperXCMD.h" and */
```

```
/* XCMDGlue.c. */
/*
 * To Build:
 *
 * (1) Create a project using*/
 * this file as well as the */
 * XCMD.Glue.c file. (Set */
 * project type to XCMD (or */
 * XFCN) from the Project menu.*/
 *
 * (2) Bring the project up to */
 * date.
 *
 * (3) Build Code Resource. */
 *
 * (4) Use ResEdit to copy the */
 * resource to your stack. */
*****/
```

```
#include <MacTypes.h>
#include <OSUtil.h>
#include <MemoryMgr.h>
#include <FileMgr.h>
#include <ResourceMgr.h>
#include "HyperXCMD.h"
```

```
pascal void main( paramPtr )
    XCmdblockPtr paramPtr;
{
    char theString[256]; /* A "Pascal" String */

    theString[0] = '\0x0B'; /* Remember, static*/
    theString[1] = 'H'; /* strings are placed*/
    theString[2] = 'E'; /* in the global pool*/
    theString[3] = 'L'; /* CODE resources such */
    theString[4] = 'L'; /* as XCMDs and XFCNS*/
    theString[5] = '0'; /* don't have access to */
    theString[6] = ' '; /* globals so you have*/
    theString[7] = 'W'; /* to discretely set the*/
    theString[8] = '0'; /* string's value */
    theString[9] = 'R';
    theString[10] = 'L';
    theString[11] = 'D';

    /* A sample callback example */
    paramPtr->returnValue = PasToZero( paramPtr, &theString );
}
```

Listing 2:

```
/* File: HyperXCMD.h */
/*
 * Interface for standard */
 * HyperCard callback routines. */
/*
 * Based on original work by */
 * Dan Winkler of Apple Computer */
/*
*****/

typedef struct Str31 {
    char data[32];
} Str31;
typedef Str31 * Str31Ptr;

typedef struct XCmdblock {
    short paramCount;
    Handle params[16];
    Handle returnValue;
    BooleanpassFlag;

    void (*entryPoint)();
}
```

```

    short request;
    short result;
    long inArgs[8];
    long outArgs[4];
} XCmdBlock;
typedef XCmdBlock *XCmdBlockPtr;

/* Callback codes */
#define xresSucc      0
#define xresFail     1
#define xresNotImp    2

/* Callback request codes */
#define xreqSendCardMessage 1
#define xreqEvalExpr      2
#define xreqStringLength   3
#define xreqStringMatch    4

#define xreqZeroBytes6
#define xreqPasToZero      7
#define xreqZeroToPas      8
#define xreqStrToLong      9
#define xreqStrToNum       10
#define xreqStrToBool      11
#define xreqStrToExt       12
#define xreqLongToStr      13
#define xreqNumToStr       14
#define xreqNumToHex       15
#define xreqBoolToStr      16
#define xreqExtToStr       17
#define xreqGetGlobal      18
#define xreqSetGlobal      19
#define xreqGetFieldByName 20
#define xreqGetFieldByNum  21
#define xreqGetFieldByID   22
#define xreqSetFieldByName 23
#define xreqSetFieldByNum  24
#define xreqSetFieldByID   25
#define xreqStringEqual    26
#define xreqReturnToPas    27
#define xreqScanToReturn   28
#define xreqScanToZero     29

/*
 * "Prototypes" for the Callbacks. Project
 * must include XCmdGlue.c.
 */

pascal void SendCardMessage();
pascal Handle EvalExpr();
pascal long StringLength();
pascal Ptr StringMatch();
pascal void ZeroBytes();
pascal Handle PasToZero();
pascal void ZeroToPas();
pascal long StrToLong();
pascal long StrToNum();
pascal Boolean StrToBool();
pascal void StrToExt();
pascal void LongToStr();
pascal void NumToStr();
pascal void NumToHex();
pascal void BoolToStr();
pascal void ExtToStr();
pascal Handle GetGlobal();
pascal void SetGlobal();
pascal Handle GetFieldByName();
pascal Handle GetFieldByNum();
pascal Handle GetFieldByID();
pascal void SetFieldByName();
pascal void SetFieldByNum();
pascal void SetFieldByID();
pascal Boolean StringEqual();

```

```

pascal void ReturnToPas();
pascal void ScanToReturn();
pascal void ScanToZero();

```

Listing 3:

```

/*****
 * File: XCMDGlue.c
 */
/*
 * Callback routines for XCMDs
 * and XFCNs. This file should
 * be included in your project
 */
/*
 * Based on original work by
 * Dan Winkler of Apple Computer
 */
*****/

#include <MacTypes.h>
#include <OSUtil.h>
#include <MemoryMgr.h>
#include <FileMgr.h>
#include <ResourceMgr.h>
#include "HyperXCmd.h"
#include <math.h>

pascal void SendCardMessage(paramPtr, msg)
    XCmdBlockPtr paramPtr;
    StringPtr msg;
/*****
 * Send a message back to
 * hypercard. The input message
 * is a Pascal String
 *****/
{
    paramPtr->inArgs[0] = (long)msg;
    paramPtr->request = xreqSendCardMessage;
    CallPascal( paramPtr->entryPoint );
}

pascal Handle EvalExpr(paramPtr, expr)
    XCmdBlockPtr paramPtr;
    StringPtr expr;
/*****
 * Evaluate a Hypertalk expression
 * returning the result as a "C"
 * string
 *****/
{
    paramPtr->inArgs[0] = (long)expr;
    paramPtr->request = xreqEvalExpr;
    CallPascal( paramPtr->entryPoint );
    return (Handle)paramPtr->outArgs[0];
}

pascal long StringLength(paramPtr, strPtr)
    XCmdBlockPtr paramPtr;
    StringPtr strPtr;
/*****
 * Counts the number of
 * characters in the input
 * string from StrPtr to end
 * of string (zero byte)
 *****/
{
    paramPtr->inArgs[0] = (long)strPtr;
    paramPtr->request = xreqStringLength;
    CallPascal( paramPtr->entryPoint );
    return (long)paramPtr->outArgs[0];
}

pascal Ptr StringMatch(paramPtr, pattern, target)
    XCmdBlockPtr paramPtr;
    StringPtr pattern;

```

```

Ptr      target;
/*****
* Case-insensitive match
* for pattern anywhere in
* target,
*
* Returns a pointer to first
* character of the first match,
* in target or NIL if no match
* found. pattern is a Pascal string,
* and target is a zero-terminated string.
*****/
(
    paramPtr->inArgs[0] = (long)pattern;
    paramPtr->inArgs[1] = (long)target;
    paramPtr->request = xreqStringMatch;
    CallPascal( paramPtr->entryPoint );
    return (Ptr)paramPtr->outArgs[0];
)

pascal void ZeroBytes(paramPtr,dstPtr,longCount)
    XCmdBlockPtr paramPtr;
    Ptr      dstPtr;
    long     longCount;
/*****
* Clear memory starting at destPtr
* through destPtr+longCount
*****/
(
    paramPtr->inArgs[0] = (long)dstPtr;
    paramPtr->inArgs[1] = longCount;
    paramPtr->request = xreqZeroBytes;
    CallPascal( paramPtr->entryPoint );
)

pascal Handle PasToZero(paramPtr,pasStr)
    XCmdBlockPtr paramPtr;
    StringPtr  pasStr;
/*****
* Convert a Pascal string (STR255)
* to a zero-terminated string.
* Returns a handle to a zero-terminated
* string. The caller must dispose the handle.
*
* Useful for setting the result or
* an argument you send from
* an XCMD to HyperTalk.
*****/
(
    paramPtr->inArgs[0] = (long)pasStr;
    paramPtr->request = xreqPasToZero;
    CallPascal( paramPtr->entryPoint );
    return (Handle)paramPtr->outArgs[0];
)

pascal void ZeroToPas(paramPtr,zeroStr,pasStr)
    XCmdBlockPtr paramPtr;
    char      *zeroStr;
    StringPtr  pasStr;
/*****
* Copy the zero-terminated
* string into the Pascal String.
*
* You create the Pascal string
* and pass it by reference.
*
*****/
(
    paramPtr->inArgs[0] = (long)zeroStr;
    paramPtr->inArgs[1] = (long)pasStr;
    paramPtr->request = xreqZeroToPas;
    CallPascal( paramPtr->entryPoint );
)

```

```

pascal long StrToLong(paramPtr,strPtr)
    XCmdBlockPtr paramPtr;
    Str31      *strPtr;
/*****
* Convert a string of ASCII
* characters to an unsigned
* long integer.
*****/
(
    paramPtr->inArgs[0] = (long)strPtr;
    paramPtr->request = xreqStrToLong;
    CallPascal( paramPtr->entryPoint );
    return (long)paramPtr->outArgs[0];
)

pascal long StrToNum(paramPtr,str)
    XCmdBlockPtr paramPtr;
    Str31      *str;
/*****
* Convert a string of ASCII
* characters to a signed
* long integer.
*****/
(
    paramPtr->inArgs[0] = (long)str;
    paramPtr->request = xreqStrToNum;
    CallPascal( paramPtr->entryPoint );
    return paramPtr->outArgs[0];
)

pascal Boolean StrToBool(paramPtr,str)
    XCmdBlockPtr paramPtr;
    Str31      *str;
/*****
* Convert the Pascal strings
* 'true' and 'false' to booleans.
*****/
(
    paramPtr->inArgs[0] = (long)str;
    paramPtr->request = xreqStrToBool;
    CallPascal( paramPtr->entryPoint );
    return (Boolean)paramPtr->outArgs[0];
)

pascal void StrToExt(paramPtr,str,myext)
    XCmdBlockPtr paramPtr;
    Str31      *str;
    long      *myext;
/*****
* Convert a string of ASCII digits
* to an extended long integer.
*
* The return value is passed
* by reference and you must
* allocate the space before
* calling this routine.
*****/
(
    paramPtr->inArgs[0] = (long)str;
    paramPtr->inArgs[1] = (long)myext;
    paramPtr->request = xreqStrToExt;
    CallPascal( paramPtr->entryPoint );
)

pascal void LongToStr(paramPtr,posNum,mystStr)
    XCmdBlockPtr paramPtr;
    long      posNum;
    Str31      *mystStr;
/*****
* Convert an unsigned long integer
* to a pascal string representation
* Useful for sending numbers back
* to Hypercard.
*

```

```

* You create mystr and pass
* it by reference.
*****/
(
    paramPtr->inArgs[0] = (long)posNum;
    paramPtr->inArgs[1] = (long)mystr;
    paramPtr->request = xreqLongToStr;
    CallPascal( paramPtr->entryPoint );
)

pascal void NumToStr(paramPtr,num,mystr)
    XCmdBlockPtr paramPtr;
    long    num;
    Str31    *mystr;
/*****
* Convert a signed long integer
* to a pascal string representation
* Useful for sending numbers back
* to Hypercard.
*
* You create mystr and pass
* it by reference.
*****/
(
    paramPtr->inArgs[0] = num;
    paramPtr->inArgs[1] = (long)mystr;
    paramPtr->request = xreqNumToStr;
    CallPascal( paramPtr->entryPoint );
)

pascal void NumToHex(paramPtr,num,nDigits,mystr)
    XCmdBlockPtr paramPtr;
    long    num;
    short    nDigits;
    Str31    *mystr;
/*****
* Convert an unsigned long integer
* to a hexadecimal number and put it
* into a Pascal string.
*
* The "output" string is passed
* by reference.
*****/
(
    paramPtr->inArgs[0] = num;
    paramPtr->inArgs[1] = nDigits;
    paramPtr->inArgs[2] = (long)mystr;
    paramPtr->request = xreqNumToHex;
    CallPascal( paramPtr->entryPoint );
)

pascal void BoolToStr(paramPtr,bool,mystr)
    XCmdBlockPtr paramPtr;
    Boolean    bool;
    Str31    *mystr;
/*****
* Convert a boolean to
* 'true' or 'false'.
*
* The "output" string is passed
* by reference.
*****/
(
    paramPtr->inArgs[0] = (long)bool;
    paramPtr->inArgs[1] = (long)mystr;
    paramPtr->request = xreqBoolToStr;
    CallPascal( paramPtr->entryPoint );
)

pascal void ExtToStr( paramPtr, myext, mystr)
    XCmdBlockPtr paramPtr;
    char    *myext;
    Str31    *mystr;
/*****

```

```

* Convert an extended long
* to its string representation
*
* The "output" string is passed
* by reference.
*****/
(
    paramPtr->inArgs[0] = (long)myext;
    paramPtr->inArgs[1] = (long)mystr;
    paramPtr->request = xreqExtToStr;
    CallPascal( paramPtr->entryPoint );
)

pascal Handle GetGlobal(paramPtr,globName)
    XCmdBlockPtr paramPtr;
    StringPtr    globName;
/*****
* Return a handle to a zero-terminated
* string containing the value of
* the specified HyperTalk global variable.
*****/
(
    paramPtr->inArgs[0] = (long)globName;
    paramPtr->request = xreqGetGlobal;
    CallPascal( paramPtr->entryPoint );
    return (Handle)paramPtr->outArgs[0];
)

pascal void SetGlobal(paramPtr,globName,globValue)
    XCmdBlockPtr paramPtr;
    StringPtr    globName;
    Handle    globValue;
/*****
* Set the value of the specified
* HyperTalk global variable to be
* the zero-terminated string in globValue.
* The contents of globValue
* are copied, you dispose the
* handle
*****/
(
    paramPtr->inArgs[0] = (long)globName;
    paramPtr->inArgs[1] = (long)globValue;
    paramPtr->request = xreqSetGlobal;
    CallPascal( paramPtr->entryPoint );
)

pascal Handle GetFieldByName(paramPtr,cardFieldFlag,fieldName)
    XCmdBlockPtr paramPtr;
    Boolean    cardFieldFlag;
    StringPtr    fieldName;
/*****
* Return a handle to a zero-terminated
* string containing the value of
* field fieldName on the current
* card. You must dispose the handle.
*
* Set cardFieldFlag to true if
* you want the contents of a card
* field or to false if you want
* the contents of a bgnd field
*****/
(
    paramPtr->inArgs[0] = (long)cardFieldFlag;
    paramPtr->inArgs[1] = (long)fieldName;
    paramPtr->request = xreqGetFieldByName;
    CallPascal( paramPtr->entryPoint );
    return (Handle)paramPtr->outArgs[0];
)

pascal Handle GetFieldByNum(paramPtr,cardFieldFlag,fieldNum)
    XCmdBlockPtr paramPtr;
    Boolean    cardFieldFlag;
    short    fieldNum;

```

```

/*****
* Returns a copy of the contents of the field whose number is
* fieldnum on the current card.
* You dispose the handle when you are done.
*****/
(
    paramPtr->inArgs[0] = (long)cardFieldFlag;
    paramPtr->inArgs[1] = fieldNum;
    paramPtr->request = xreqGetFieldByNum;
    CallPascal( paramPtr->entryPoint );
    return (Handle)paramPtr->outArgs[0];
)

pascal Handle GetFieldByID(paramPtr,cardFieldFlag,fieldID)
    XCmdBlockPtr paramPtr;
    Boolean      cardFieldFlag;
    short        fieldID;
/*****
* Returns a copy of the contents of the field whose id is
* fieldID on the current card.
* You dispose the handle when you are done.
*****/
(
    paramPtr->inArgs[0] = (long)cardFieldFlag;
    paramPtr->inArgs[1] = fieldID;
    paramPtr->request = xreqGetFieldByID;
    CallPascal( paramPtr->entryPoint );
    return (Handle)paramPtr->outArgs[0];
)

pascal void
SetFieldByName(paramPtr,cardFieldFlag,fieldName,fieldVal)
    XCmdBlockPtr paramPtr;
    Boolean      cardFieldFlag;
    StringPtr    fieldName;
    Handle       fieldVal;
/*****
* Set the value of the field whose name is fieldName on
* the current card.
* You dispose the handle when you are done.
*****/
(
    paramPtr->inArgs[0] = (long)cardFieldFlag;
    paramPtr->inArgs[1] = (long)fieldName;
    paramPtr->inArgs[2] = (long)fieldVal;
    paramPtr->request = xreqSetFieldByName;
    CallPascal( paramPtr->entryPoint );
)

pascal void
SetFieldByNum(paramPtr,cardFieldFlag,fieldNum,fieldVal)
    XCmdBlockPtr paramPtr;
    Boolean      cardFieldFlag;
    short        fieldNum;
    Handle       fieldVal;
/*****
* Set the value of the field whose number is fieldnum on
* the current card.
* You dispose the handle when you are done.
*****/
(
    paramPtr->inArgs[0] = (long)cardFieldFlag;
    paramPtr->inArgs[1] = fieldNum;
    paramPtr->inArgs[2] = (long)fieldVal;
    paramPtr->request = xreqSetFieldByNum;
    CallPascal( paramPtr->entryPoint );
)

pascal void
SetFieldByID(paramPtr,cardFieldFlag,fieldID,fieldVal)
    XCmdBlockPtr paramPtr;
    Boolean      cardFieldFlag;
    short        fieldID;
    Handle       fieldVal;

```

```

/*****
* Set the value of the field whose id is fieldID on
* the current card.
* You dispose the handle when
* you are done.
*****/
(
    paramPtr->inArgs[0] = (long)cardFieldFlag;
    paramPtr->inArgs[1] = fieldID;
    paramPtr->inArgs[2] = (long)fieldVal;
    paramPtr->request = xreqSetFieldByID;
    CallPascal( paramPtr->entryPoint );
)

pascal Boolean StringEqual(paramPtr,str1,str2)
    XCmdBlockPtr paramPtr;
    Str31        *str1;
    Str31        *str2;
/*****
* Returns true if the strings match, false otherwise.
* Compare is case insensitive
*****/
(
    paramPtr->inArgs[0] = (long)str1;
    paramPtr->inArgs[1] = (long)str2;
    paramPtr->request = xreqStringEqual;
    CallPascal( paramPtr->entryPoint );
    return (Boolean)paramPtr->outArgs[0];
)

pascal void ReturnToPas(paramPtr,zeroStr,pasStr)
    XCmdBlockPtr paramPtr;
    Ptr          zeroStr;
    StringPtr     pasStr;
/*****
* Collect characters from zeroStr
* to the next carriage Return and return
* them in the Pascal string pasStr.
* If no Return found, collect chars
* until the end of the string (zero)
*****/
(
    paramPtr->inArgs[0] = (long)zeroStr;
    paramPtr->inArgs[1] = (long)pasStr;
    paramPtr->request = xreqReturnToPas;
    CallPascal( paramPtr->entryPoint );
)

pascal void ScanToReturn(paramPtr,scanHnd1)
    XCmdBlockPtr paramPtr;
    Ptr          *scanHnd1;
/*****
* Position the pointer, scanPtr, at a Return character
* or a zero byte.
*****/
(
    paramPtr->inArgs[0] = (long)scanHnd1;
    paramPtr->request = xreqScanToReturn;
    CallPascal( paramPtr->entryPoint );
)

pascal void ScanToZero(paramPtr,scanHnd1)
    XCmdBlockPtr paramPtr;
    Ptr          *scanHnd1;
/*****
* Position the pointer, scanPtr,
* at a zero byte.
*****/
(
    paramPtr->inArgs[0] = (long)scanHnd1;
    paramPtr->request = xreqScanToZero;
    CallPascal( paramPtr->entryPoint );
)

```



HyperChat™

XCMD Corner: XCMD Libraries



Donald Koscheka
Arthur Young & Co.
MacTutor Contributing Editor
Volume 5, Number 6

Modularity and Coupling

As you to explore XCMDs, you should continue to discover new and interesting properties of these routines. Two properties are particularly noteworthy because they lead to code that is usable outside of Hypercard. They are strong modularity and weak coupling.

You determine the modularity of a routine by counting the number of tasks it performs. Strongly modular routines handle one task. If you were writing a spreadsheet program, you would most likely separate the routine that calculates the value of a cell from the routine that displays that value. Combining these functions results in weak modularity. Weakly modular code is hard to debug because you can easily lose track of what the routine is doing at a given point.

Coupling describes how the program interacts with the outside world. Weakly coupled routines operate only on data passed as formal parameters. Weak coupling eliminates harmful side-effects caused by relying on global memory. Weakly coupled programs force you to think over how and when to pass data back and forth.

Well written XCMDs exhibit both strong modularity and weak coupling. Most XCMDs perform one task (strongly modular XCMDs), typically to add a missing feature to Hypertalk. XCMDs interface to Hypertalk using a very strict protocol, passing parameters via the command block record. This protocol coerces weakly coupling in your code.

Software Breadboard

Strong modularity and weak coupling suggest that you can create libraries of useful routines for use both in Hypercard and in programs of your own design. Think of Hypercard as a "Software Breadboard", a test platform that you plug code into for checkout and debugging. Once working, the routine can be dropped into any application.

An example of a set of routines that play well in this scenario are the calls to the File Manager. Every (non-trivial) application needs to call on the file manager to open a file, access information in the file and then close it. I decided to build my case around accessing a file because Hypercard itself doesn't handle files "according to Hoyle". Hypercard violates two precepts of the user interface guidelines when opening a file: (1) The user is required to know the full pathname of a file if it is not in the current working directory and (2) the OpenFile command performs an unexpected action; if it cannot find the file, it creates it!

In order to properly open a file, you first present the user with a dialog that contains the list of files in the current working directory. This dialog allows the user to flip from drive to drive as well as change directories. The standard file package provides

you with the "vanilla" get file dialog. When the user selects a file and clicks the "Open" button, your code queries the reply record to discover the file's name, working directory id and file type.

The working directory id pinpoints which folder contains the document. If you made a killing on Macintosh software and have been away on some deserted island for the past several years, the working directory assumes the role traditionally held by the volume reference number.

Given the file name and working directory, you can reconstruct and return the full pathname to Hypercard so that it can be used by Hypercard's "open" command (pathname reconstruction is the subject of a future article). However, it's just as easy to open the file ourselves to ensure that it opens in whatever manner is expected on the Macintosh. The XCMD that opens the file is left as an exercise.

GetFileName XFCN

The XFCN, GetFileName (Listing 1), displays strong modularity - its sole task is to get the name of a file from the user. By relying only on data passed to it formally, it also demonstrates weak coupling. This XFCN provides a general purpose technique for getting the name of a file from the user. GetFileName calls on a routine Called GetFileNameToLoad in HyperUtils.c (listing 2). The prototypes for HyperUtils are found in HyperUtils.h (listing 3). HyperUtils contains a set of re-usable utilities that do not presume the existence of Hypertalk. These routines aren't encumbered with the XCmdBlkPtr interface.

GetFileNameToLoad accepts as its inputs a list of file types (up to 4) to filter out in the open dialog. If you were writing a text import routine, you might want to present the user with only files of type 'TEXT'. From Hypercard, you would then invoke GetFileName as:

```
Put GetFileName( "TEXT" ) into it;      IF it is not
empty then - User selected a file      put item 1 of it
into FileName                          put item 2 of it into FolderID
end if
```

With little effort, we've created a re-usable, general purpose XCMD. Pass the name and the FolderID to any File Manager call that requires the file's name and volume reference number (aka working directory id). I will call upon this XCMD in future editions of this column so you'll have ample opportunity to see it action.

An obvious addition to this list an xcmd that asks the user for the name of a file to save by calling SFPutFile. This is a simple modification to GetFileNameToLoad so I'll leave it as yet another exercise.

After creating enough XCMDs, you'll soon view them as

general purpose routines with a specific interface (XCm dBkPtr) to HyperTalk. With this in mind, you should have no trouble creating XCMD libraries that can be used in other applications. Get in the habit of separating the Hypertalk interface, including any callbacks, from the action code. The XCMD itself then becomes an interface between HyperTalk and your code.

Listing 1: GetFileName.c

```

/*****
/* File: GetFileName.c */
/*
/* Using Standard File Package */
/* query the user for the name */
/* of a file to open, and return */
/* the name along with the */
/* working directory id of the */
/* file. */
/* Parameters: */
/* paramCnt = number of types */
/* params[0..3] = the types to */
/* filter for (see Inside Mac. */
/* I-523 for details */
/*
/* To Build: */
/* (1) Create a project using */
/* this file as well as the */
/* XCMD.Glue.c file. (Set */
/* project type to XCMD (or */
/* XFCN) from the Project menu. */
/*
/* (2) Bring the project up to */
/* date. */
/* (3) Build Code Resource. */
/* (4) Use ResEdit to copy the */
/* resource to your stack. */
*****/

#include <MacTypes.h>
#include <OSUtil.h>
#include <MemoryMgr.h>
#include <FileMgr.h>
#include <ResourceMgr.h>
#include <pascal.h>
#include <strings.h>
#include "HyperXCmd.h"
#include "HyperUtils.h"

pascal void main( paramPtr )
    XCm dBkPtr paramPtr;
{
    short FileWID;
    short numTypes;
    short i;
    SFTYPEList tps;
    char FileName[256];
    char WIDString[32];
    char comma[2];

    if( !paramPtr->paramCount )
        numTypes = -1; /* select all since no type specified */
    else{
        numTypes = paramPtr->paramCount;
        for( i = 0; i < numTypes; i++ )
            BlockMove( *(paramPtr->params[i]), &tps[i], 4L );
    }

    *FileName = '\0';

    if( GetFileNameToOpen( tps, numTypes, FileName, &FileWID )
){

```

```

    NumToStr( paramPtr, (long)FileWID, &WIDString );
    PtoCstr( WIDString );

    comma[0] = ','; /* for you MPW folk */
    comma[1] = '\0';

    strcat( FileName, comma );
    strcat( FileName, WIDString );
    CtoPstr( FileName );
}

paramPtr->returnValue = PasToZero( paramPtr, FileName );
}

```

Listing 2: HyperUtils.c

```

/*****
/* HyperUtils.c */
/* A collection of useful */
/* routines... */
*****/

#include <MacTypes.h>
#include <OSUtil.h>
#include <MemoryMgr.h>
#include <FileMgr.h>
#include <ResourceMgr.h>
#include <StdFilePkg.h>
#include "HyperXCmd.h"
#include "HyperUtils.h"

void CenterWindow( wptr )
    WindowPtr wptr;
/*****
* Center a window in the current
* screen port. Note: Does not
* attempt to work with multi-screen
* systems.
*
* This code is inspired by a
* similar routine written by Steve
* Maller in MPW Pascal. Thanks Steve.
*****/
{
    short hWindSize = wptr->portRect.right - wptr->portRect.left;
    short vWindSize = wptr->portRect.bottom - wptr->portRect.top;
    short hSize = wptr->portBits.bounds.right - wptr->portBits.bounds.left;
    short vSize = wptr->portBits.bounds.bottom - wptr->portBits.bounds.top;

    MoveWindow( wptr,
        ( hSize - hWindSize ) / 2,
        ( vSize - vWindSize + 20 ) / 2,
        false
    );
}

void Concat( str1, str2 )
    char *str1;
    char *str2;
/*****
* Append string 2 to the end of
* string 1. Both strings are
* pascal-format strings.
*
* str1 must be large enough to hold
* the new string and is assumed to
* be of Type Str255 (a pascal string)
*****/
{
    short len1 = *str1; /***number of chars in str 1***/
    short len2 = *str2++; /***number of chars in str 2***/
    char *temp; /*** string pointer ***/

    *str1 += len2 + 1; /*** add sizes to get new size***/

```

```

temp = str1 + len1 + 1; /** move to end of str 1***/
while( len2 ){
    *temp++ = *str2++; /** add char to temp & move***/
    -len2; /** until all characters are added***/
}
}

```

```

void CopyPStr( pStr1, pStr2 )
char *pStr1;
char *pStr2;
/*****
* Copy the contents of pstr1 into
* pstr2. The strings are assumed
* to be of type STR255 (length byte
* precedes data
*
*****/
{ short i;
  char *tstr;

  tstr = pStr2;

  for( i = 0; i <= *pStr1; i++ )
      *tstr++ = *pStr1++;
}

```

```

short GetFileNameToOpen( typs, typCnt, theName, theWDID )
SFTypeList typs;
short typCnt;
char *theName;
short *theWDID;
/*****
* Invokes SFOpenFile to query the
* user for the name of a file to
* open.
*
* In: List of types of files to
* filter for (up to 4)
* Out: fileName if picked in theName
* working directory in theWDID
* nil otherwise
* the file's volum ref num.
* ( Note that the space for the
* string must be allocated by the
* caller).
*****/
{
    Point where;
    char prompt[1];
    SFReply reply;
    GrafPort *oldPort;
    WindowPtr dlogID;

```

```

prompt[0] = '\0';

```

```

/** Get and put up the standard file ***/
/** dialog. You will only see the file***/
/** types that you filtered for. If ***/
/** you filtered for no files, then ***/
/** all files will display ***/

```

```

GetPort( &oldPort );
dlogID = GetNewDialog( (short)getDlgID, (Ptr)NIL,
(Ptr)UPFRONT );

```

```

SetPort( dlogID );
CenterWindow( dlogID );
where.h = dlogID->portRect.left;
where.v = dlogID->portRect.top;
LocalToGlobal( &where );

```

```

SFGetFile( where, prompt, (Ptr)NIL, typCnt, typs, (Ptr)NIL,
&reply );

```

```

DisposDialog( dlogID );
SetPort( oldPort );

```

```

/** If the user selected a file, let's ***/
/** get the information about it ***/

```

```

if (reply.good){
    *theWDID = reply.vRefNum;
    PtoCstr( (char *)&reply.fName );
    strcpy( theName, &reply.fName );
}
return( reply.good );
}

```

Listing 3: HyperUtils.H

```

/*****
* HyperUtils.H
* Header file for HyperUtils.c*
* routines...
*****/

```

```

#define NIL 0L
#define UPFRONT -1L

```

```

void CenterWindow( WindowPtr wptr );
void Concat( char * str1, char * str2 );
void CopyPStr( char * pStr1, char * pStr2 );
short GetFileNameToOpen( SFTypeList typs, short typcnt, char
*theName, short *theWDID);

```



HyperChat™

HyperArrays



Fred Stauder
Scott Vore
Indianapolis, IN
Volume 5, Number 6

on HyperChat

HyperMedicine: Using Hypercard to Solve Problems

This month we will look at ways people have solved problems in the medical field (my origin) using Hypercard. One of the reasons Hypercard is so popular in the medical field is that after spending so many years learning to be a doctor people feel that investing more to be a computer programmer is not justified. There is very much a "We want solutions now!" mentality.

The first problem was how do you get the patient more involved in his/her case decision making process without tying up too much of your surgeon's time? Harold Lyon from Dartmouth Medical School used Hypercard and a laserdisc to form an interactive decision making tool. The topic is "Prostatectomy or Watchful Waiting". The disc shows the risks and rewards of having the operation. Patients are interviewed and even doctors that have been patients. It gives you all the pros and cons of having the operation. The other purpose of the laserdisc is to gather data about patients and follow up treatments. This program that they have created has legal, ethical, educational, and research implications. It also protects the doctor by documenting the patients informed consent.

Harold Lyon gives some useful tips in preparing a videodisc>

- Always put SMPTE on the source tape.
- Flowchart the disc
- Use the same microphones at the same distance to maintain sound quality
- Train the speakers not to speak at the same time
- Work with one studio and film group
- Use small organized teams
- Never try to insert single words in audio
- Reach final script consensus prior to off-line edit
- Use colorbars on original source tapes
- Keep good track of script versions for team
- Plan the important details first
- Prepare more before on-line edit
- The video editor must be part of the team not someone hired for piecemeal work
- Consider carefully the use of one versus two camera shoots
- Ask interviewee the questions on video. Tape record these questions, play them back, and write them down. Take some head shots for cutaways. Then take the interviewer asking the same questions with a delay between each. Also take shots of interviewer sitting quietly for cutaways.
- Allow some spontaneity on line
- Film making and videodisc film making require different skills- be careful who you hire.
- Evaluate the disc before it is pressed changes are hard to make after pressing
- MacRecorder from Farralon was invaluable
- Consult experts

- Study other potential markets for your videodisc before shooting. With minor additions different applications can be made.

These tips are typical of the type of problems you will encounter when you make a videodisc.

The second interesting application is an article on XCMD's written By Scott Vore MD. He is an anesthesiologist who wanted to track patients blood pressure, heart rate, drugs administered etc. He decided to do it in Hypercard however he found that he had to keep track of an array of data. So he wrote an XCMD called Hyperarray to do just that.

If you hear of any interesting applications Hypercard has been put to send them to us.

Send articles ideas, comments etc to Applelink: STAUDER

end HyperChat

HYPER ARRAYS

[Scott's background is entirely self taught but he has been at it for 3 plus years now]

First of all, before you read any further I feel compelled to make the following disclaimers:

I am not, never have been, never plan to be a 'real' programmer. I'm totally self taught, thanks in most part to a handful of books and one excellent journal (we know which one)-consequently my programming 'style', if 'style' can describe it, contains bits and pieces of code examples I've seen elsewhere and often; in my haste to get things to work, I'll leave in bits and pieces of code that serve no real purpose but should be removed if programming correctness were to be maintained. I'm sorry if this has happened here- with more time I could make it all prettier but I hope I've given someone enough to work with that they too can write and make things neat. My only motivation has been to make this machine work for me, (with no formal training I have never felt competent to try and teach others). In that process, however I have learned a few things and some of those things seem important (I guess I'll let the editors decide just how important). Oh well, on with the show...

In my programming experience with Hypercard, I have never ceased to be amazed at the power, elegance, and simplicity inherent in this product. Every now and again, however, we all run up against 'the wall' and are forced to either change our approach or find another method to solving a particular problem.

Currently I am writing a stack that will be used in an Operating Room environment to allow an Anesthesiologist to record various pieces of data at specific time intervals throughout a case. Typically, the blood pressure, heart rate, and various other measurements are kept track of throughout a case so that the Anesthesiologist can keep track of the trend in these parameters as time passes, additional drugs are administered, and as the surgery progresses.

My particular problem was in storing the information in a

way that was accessible in a random fashion, retrievable between calls to different stacks and between calls to shutdown that stack. In other words I wanted a data structure that was 'global' in nature and that acted as an array.

Being a faithful MacTutor reader I am aware that we are being admonished to always try and solve our problems in Hypertalk before resorting to writing an XCMD, so in all fairness I must say that the XCMD described below does have a Hypertalk counterpart although somewhat more cumbersome. The example stack I have enclosed will compare the two methods.

Ok, enough of all this, what is my problem?

I wanted to create an array that was accessible throughout Hypercard. Fine. How? One method is to create a field somewhere in my stack that has each line representing a different time point and each item on that line representing a different array variable.

e.g. put 10 into line 10 of cd fld "array"

While this approach does serve its purpose well a multi-dimensional array must rely on scripting to enter the various data points with commas interspersed between items and those same commas must be stripped off as the data is retrievable- doable but messy.

My approach was to create a one dimensional array (though a multidimensional array can easily be created) as a resource, attach that resource to the stack in question and access that array with calls to an XCMD. The array is very quickly accessed, is 'global' in nature and can be accessed easily through Hypercard.

The first step consists in determining the size of the array. For the example stack I have created a 60 element array since in a time based environment 60 works out well. Now, the array Resource must be created. There are two options here- the resource can be created from the program (from the XCMD) or it can be created with Rmaker and pasted into the stack in question. I chose the latter approach (to me, it was the easiest way).

At this point decide on a Resource type too.

I chose ANES for the simple reason that I am an anesthesiologist .

The Rmaker source code is as follows:

```
ANESTH.RSRC

type ANES = GNRL
,1005
.H
0000 0000 0000 0000
0000 0000 0000 0000
0000 0000 0000 0000
0000 0000 0000 0000
0000 0000 0000 0000
0000 0000 0000 0000
0000 0000 0000 0000
0000 0000 0000 0000
0000 0000 0000 0000
0000 0000 0000 0000
0000 0000 0000 0000
0000 0000 0000 0000
0000 0000 0000 0000
0000 0000 0000 0000
0000 0000 0000 0000
```

This, obviously initializes the values to zero and sets the values in hexadecimal format though integer or any other Rmaker legal format could have been used. If the hex format is

maintained and you want to store character values then be sure and use some variation of the numtostring function to convert the character to its ASCII value and the reciprocal function to convert it back on retrieval.

After running Rmaker, the resource would exist with id 1005 and could then be pasted into the stack of your choice with Resedit. So, assuming this has all been done the next step is to write the XCMD's.

At this point knowing exactly what you want the XCMD to do will save all sorts of frustration and reworking later (trust me). So, a small digression is in order as we set up the stack and decide what it is we want to accomplish with our array.

First, and foremost, the book that everyone is talking about, Gary Bond's XCMD's for Hypercard, should be your computer side companion if you want quick access to terminology, examples and good programming style while writing your commands. Next, depending on the development system you are using open the HyperX... interface files to see the exact format that the procedures are written in since they aren't all written the same way that the examples in Gary Bond's book assumes (some pass the paramblock ptr in all procedures).

For the sake of illustration and simplicity we will set up our stack in such a way that we will (1) fill the array with 60 values from some source (a quick and dirty method will be to use Hypercard's random number generator to generate 60 values) although these values could come from a field, an ask dialog, anywhere.

(2) we will access these values in two ways

(a) randomly-just to prove it works

and (b) in sequence to fill in a graph.

(3) as an added feature we will also continually update the graph described in 2b above so that new points can be added and the old ones removed.

The above stack will require at least three but probably four XCMD's to fulfill it's requirements. The first will fill the array RSRC with values on a random access basis. The second will retrieve values from the array and the third will create a 'buffer array' that will serve to save 60 data points- in the example stack this will hold the previous 60 points that the program generated so that they can be erased as the new points are plotted. This means that an additional 'ANES' RSRC must be created with a different id to serve as a buffer- but that is simply done with Resedit(copy/paste), and it's get info menu item. The fourth XCMD will then access the buffer array and not the working array to retrieve previous 'saved' values'

Again, a slight digression is in order here, to stand back and see exactly what these items can allow us to do. In the example stack, the array is somewhat simple minded, but with a little imagination, the possibilities of such a Resource are wide. For example, the id of all cards can be kept in the array to give a 'recording' of movement throughout the stack, a record of any and all users of a stack can be kept, initialization values can be kept here, the buffer array can be huge so that it can be updated every time the working array is filled. Ok, maybe I'm the only one excited about all these possibilities, but hopefully some of you out there will be turned on too.

So, the first XCMD to write will be the one to access the array at any point and fill in the data at that point.

I have chosen to call it (in a fit of imagination and creativity-PUTDATA). The XCMD expects to receive two parameters- the time or array index and the value at that point. Below is the source

code of the PUTDATA xcmd:

```

unit putdataXcmd;
interface
  uses MemTypes, QuickDraw, OSIntf, ToolIntf, PackIntf,
  HyperXCMD, QDAccess;

  procedure putdata(ParamPtr: XCMDPtr);

implementation
  type
    timeDarray=array[0..59] of integer;
  procedure arrayrsc(ParamPtr:XCMDPtr);forward;

  procedure putdata(paramptr:xcmdptr);
  begin
    arrayrsc(paramptr);
  end;

  procedure arrayrsc(ParamPtr: XCMDPtr);
  var
    MYTIMEHAND:HANDLE;
    REFNUM:INTEGER;
    TIMEARRAY:TIMEDARRAY;
    temphandle:handle;
    tempstr:str255;
    horiz,vert:longint;
    (***** get points *****);
  procedure getpoints( paramptr :xcmdptr;var
  horiz,vert:longint);
  begin zerotopas(paramptr,paramptr^.params[1]^,tempstr);
    horiz:=strtonum(paramptr,tempstr);
    zerotopas(paramptr,paramptr^.params[2]^,tempstr);
    vert:=strtonum(paramptr,tempstr);
  end;
  (*****);
  begin
    mytimehand :=(getresource('ANES',1005));
    hlock(mytimehand);
    blockmove(mytimehand^,@timearray, sizeof(timearray));
    getpoints(paramptr,horiz,vert);
    if horiz > 59 then horiz := 59;
    timearray[horiz] := vert;
    blockmove(@timearray,mytimehand^,sizeof(timearray));
    refnum:=currenfile;
    changedresource(mytimehand);
    writeresource(mytimehand);
    releaseresource(mytimehand);
  end;
end.

```

The XCMD is lacking in many things, I didn't error check after the call to getresource so the user will have no way of knowing whether the call was successful or not. Also there is no check on the number of parameters passed to the XCMD- sheer laziness I guess. But, assuming that the user is the one writing the XCMD it should be his/her decision as to whether or not to make this XCMD general enough for the public at large or specific to his/her application (the latter approach implies that the programmer using the XCMD would be versed in it's parameter requirements and would pass the correct values).

The second XCMD is very similar to the first, and in fact the two could be combined in such a way that a parameter check would tell if two parameters were passed indicating the user wanted to set the array value of param[1] to param[2] or if one parameter were passed indicating the user desired to retrieve the array value of param[1]. Again I've stuck with my original version of writing these things, not because they are better, but because they serve to indicate in some small sense, the evolution that these XCMD's have undergone and they serve to suggest a number of ways to improve upon and write better XCMD's. The second XCMD is titled 'GetData' and expects one argument as a parameter- the value of the array index to be read. The result is passed back to Hypercard in the result field but could easily be placed in a global variable or field.(See Gary Bond's book for the gory details).

```

unit getdataXcmd;
interface
  uses MemTypes, QuickDraw, OSIntf, ToolIntf, PackIntf,
  HyperXCMD, QDAccess;
  procedure getdata(ParamPtr: XCMDPtr);
implementation
  type
    timeDarray=array[0..59] of integer;
  procedure arrayrsc(ParamPtr: XCMDPtr);forward;
  procedure getdata(paramptr:xcmdptr);
  begin
    arrayrsc(paramptr);
  end;
  procedure arrayrsc(ParamPtr: XCMDPtr);
  var
    MYTIMEHAND:HANDLE;
    REFNUM:INTEGER;
    TIMEARRAY:TIMEDARRAY;
    temphandle:handle;
    a:integer;
    tempstr:str255;
    horiz:longint;
  procedure getpoints(Paramptr:xcmdptr,var num:longint);
  var tempstr1:str255;
  begin
    zerotopas(paramptr,paramptr^.params[1]^,tempstr1);
    horiz:=strtonum(paramptr,tempstr1);
  end;
  begin
    mytimehand :=(getresource('ANES',1005));
    hlock(mytimehand);
    blockmove(mytimehand^,@timearray,sizeof(timearray));
    getpoints(paramptr,horiz);
    a:=timearray[horiz];
    longtostr(paramptr,a,tempstr);
    paramptr^.returnvalue:= pastozero(paramptr,tempstr);
    hunlock(mytimehand);
    releaseresource(mytimehand);
  end;
end.

```

Again, no error checking on getresource or number of params.

By themselves, or combined as a single XCMD, these examples will allow random access to a 60 point array from within Hypercard. The array size can easily be changed in the above examples and, in the initial resource definition, to allow any size array to be used—though I think the 32K limit also applies to resources.

The third XCMD I've written is used to set up a buffer so that the initial array can continue to be updated while the 'old values'

are saved. An example of it's use is in the example stack and has been described above. It's call 'DATABUFF' and requires no parameters- assuming that the two array rsrc's are present in the stack file and that their id's are known. Clearly this approach does not allow much room for error but, with careful planning, should not be much of a problem.

```

unit databuffXcmd;
interface
  uses MemTypes, QuickDraw, OSIntf, ToolIntf, PackIntf,
  HyperXCMD, QDAccess;

  procedure databuff(ParamPtr: XCMDPtr);

implementation
type
  timeDArray=array[0..59] of integer;
procedure arrayrsrc(ParamPtr: XCMDPtr);forward;
procedure databuff(paramptr:xcmdptr);
begin
  arrayrsrc(paramptr);
end;

procedure arrayrsrc(ParamPtr: XCMDPtr);
var
  mytimehand, buffarrayH:HANDLE;
  REFNUM: INTEGER;
  TIMEARRAY, buffarray:TIMEDARRAY;
  tempHandle:handle;
  tempstr:str255;
  horiz,vert:longint;
  numparams :integer;

begin
  mytimehand := (getresource('ANES',1005));
  hlock(mytimehand); BLOCKMOVE(MYTIMEHAND^,@timearray,
                                SIZEOF(TIMEARRAY));
  releaseresource(mytimehand);
  buffarrayH:=(getresource('ANES',1010));
  hlock(buffarrayH);
  blockmove(buffarrayH^,@buffarray, sizeof(buffarray));
  buffarray := timearray;
  blockmove(@buffarray,buffarrayH^, sizeof(buffarray));

  REFNUM:=CURRESFILE;
  changedresource(buffarrayH);
  writeresource(buffarrayH);
  hunlock(buffarrayH);
  releaseresource(buffarrayH);
end;
end.

```

The fourth XCMD must be used with the databuff XCMD. It is actually a copy of the XCMD 'getdata' with the exception that it accesses the buffer array (with a different id number) so that saved values can be read back and acted upon accordingly.

```

unit getprevdataXcmd;
interface
  uses MemTypes, QuickDraw, OSIntf, ToolIntf, PackIntf,
  HyperXCMD, QDAccess;
  procedure getprevdata(ParamPtr: XCMDPtr);
implementation
type
  timeDArray=array[0..59] of integer;
procedure arrayrsrc(ParamPtr: XCMDPtr);forward;
procedure getprevdata(paramptr:xcmdptr);
begin
  arrayrsrc(paramptr);
end;

```

```

procedure arrayrsrc(ParamPtr: XCMDPtr);
var
  MYTIMEHAND:HANDLE;
  REFNUM: INTEGER;
  TIMEARRAY:TIMEDARRAY;
  tempHandle:handle;
  tempstr:str255;
  a :integer;
  vert,b:integer;
  horiz:longint;
procedure getpoints(Paramptr:xcmdptr;var;num:longint);
var tempstr1:str255;
begin
  zerotopas(paramptr,paramptr^.params[1], tempstr1);
  horiz:=strtonum(paramptr, tempstr1);
end;
begin
  mytimehand:=(getresource('ANES',1010));
  HLOCK(MYTIMEHAND);
  blockmove(mytimehand^,@timearray,sizeof(timearray));
  getpoints(paramptr,horiz);
  a:=timearray[horiz];
  longtostr(paramptr,a, tempstr);
  paramptr^.returnvalue:=pastozero(paramptr, tempstr);
  releaseresource(mytimehand);
end;
end.

```

Figure 1 is a table summarizing the syntax of the XCMD's and the id's of the array resources.

RESOURCE ID'S:

ANES 1005 -This is the working array
ANES 1010 -This is the buffer array

XCMD SYNTAX:

PUTDATA arrayindex,value

-this XCMD expects two parameters, it will place the value parameter into the array at the position arrayindex. e.g. array[1] := value;

GETDATA arrayindex

-this XCMD expects one parameter. It will return the value of the array at arrayindex in the Hypercard variable 'the result'
e.g array[arrayindex] := theresult;

DATABUFF

this XCMD has no parameters passed with it. It takes the values in the array represented by Resource 1005 (the array the user manipulates in the above two calls) and places them in the array represented by RSRC 1010.

GETPREVIOUSDATA arrayindex

-this XCMD is identical to the GETDATA XCMD in that it is passed a single parameter -arrayindex and it accesses the buffer array (id 1010) to get its' value (which is also returned in 'the result').

Figure 1

Well, that's about it for the XCMD's. There is a lot of room for improvement here and I believe a lot of room to explore. I've included the XCMD's and an example stack that demonstrates their use as well as the Hypertalk method of using an array. I apologize again for my lack of programming expertise and hope that it doesn't detract from the overall article. I would like to write another article in the near future that gives a few tricks to use the third party product "Programmer's Extender" in XCMD's in order to save even more time and effort in writing XCMD's- let me know if anyone is interested in this.

The example stack contains three buttons:

The first will demonstrate the use of a Hypertalk array in the

best way I know how - it's entitled "the old way" and basically initializes 50 lines of an invisible card field to some random number .

It's script is as follows:

```
on mouseup
  put 1 into linecount
  repeat(50)
    put random(50) into line linecount of cd fld "array"
    put linecount + 1 into linecount
  end repeat
end mouseup
```

The user can then access any point in the array with the button "get data"- it's script is as follows:

```
on mouseup
  ask "what point"
  put line it of cd fld "array"
end mouseup
```

The next card will do basically the same thing except it will use the XCMD's Putdata and Getdata.

The script of cd btn "new way" is:

```
on mouseup
  put 1 into linecount
  repeat(50)
    putdata linecount,random(50)
```

```
    put 1 + linecount into linecount
  end repeat
end mouseup
```

The script of card button "get data" is :

```
on mouseup
  ask "what point"
  put the result into pointvar
  put getdata pointvar
end mouseup
```

The final card will draw a graph of points - the horizontal coordinates will represent the array index and the vertical will represent the array index value.

It demonstrates the use of the databuff XCMD in that the graph will be continuously updated until the user presses the mouse button.

Well, that's it for my first attempt at sharing the few things I've learned about Macintosh programming. If there are questions my address is

915 N. Bolton Ave
Indpls., In. 46219.



HyperChat™

XCMD Corner: Sector Dumps



Donald Koschek
Arthur Young & Company
MacTutor Contributing Editor
Volume 5, Number 7

Reinventing the wheel often provides an opportunity to embellish on the original design. Recently, I needed a file dump utility. Several packages are available "over the counter", but I needed a utility that dumps the entire file without having to manually move from sector to sector as is the case with most commercial solutions. Moreover, I use file dump tools frequently enough that I could afford to spend the effort writing one for Hypercard.

I dump the file in a fairly typical way. Each line in the dump contains the "sector address" of the first byte on the line followed by the hexadecimal dump of 16 bytes followed by the ASCII interpretation of the data. Not all ASCII characters are "printable" so we replace non-printing characters with the ".", yielding a cleaner display.

This month's XFCN, FilePeek.c (listing 1) accepts two parameters. The first parameter is the reference number of the opened file; the second parameter is the sector you want dumped. HFS sectors are 512 bytes so filepeek returns 32 lines of 16 bytes apiece.

Dumping the entire contents of a file becomes a matter of calling FilePeek repeatedly for all the logical sectors in the file.

The first line in the dump tells us how many bytes were read in. This number can be equal to or less than a full sector. If line 1 contains less than a sector full of data, then you know that logical end of file is after the last byte read in.

The Hypertalk aficionado will realize that this code can easily be written in Hypertalk. I chose to commit my scheme to an XFCN only after discovering that the Hypertalk version was too slow for my purposes.

The dumping scheme is amazingly simple. Position the file mark at the beginning of the sector that you want to dump. If the start of the sector extends beyond end of file, do nothing and return to hypercard with a result of zero (no bytes were read in), otherwise read the sector into the buffer we allocated (buf).

The sector is presented to Hypercard as a series of hexadecimal characters. Hypercard contains a call back "NumtoHex" which converts an arbitrary run of data to a hex string. We lose portability when using a callback so you'll need to write your own hexdump algorithm or dig one up in the toolbox.

FilePeek dumps two bytes at a time (4 hex digits). This is analogous to dumping the contents of a short. We can use a pointer to short (rPtr). To dump the ascii data, we use a char pointer (cPtr) to the same data.

The current line in the dump is assembled by first displaying the "sector address" of that line. The sector address is $512 * \text{blk}$ (blk = sector number).

The sector address is delineated by the ":" character. Following the sector address, we display 16 bytes of data in hexa-

decimal format. The data is grouped into 8 words.

The hexadecimal data is followed by the ascii representation of the line. If the character falls in the range of printing characters, we display the character, otherwise print a ".". Characters below the space (0x020) are not normally considered printing characters. Likewise, Inside Macintosh indicates that no characters above \$D8 have a printable format. Using the period to represent non-printing characters results in a less-cluttered looking dump.

We need to cycle the character pointer, cPtr, twice for every cycle on rPtr (a character is half a word). We accumulate the ascii data into a separate string which we then concatenate onto the end of the hex data.

Once the current line is assembled, we stick a carriage return on the end, block move it to the output data handle and then go to the next 16 bytes in the input stream. When the outer loop completes, outdata will contain 32 lines of 16 bytes apiece, suitable for framing.

How you open and close a file from Hypercard is a matter of taste. I prefer to keep my interface to the file manager as clean as possible. This requires passing a filename and working directory id to the file manager's FSOpen call and subsequently referring to the file by the reference number returned by FSOpen. To close the file, pass the reference number to FSClose (last month I provided an XFCN that will return a file name and working directory id from the standard file package).

Listings 2 and 3 are two simple XFCNs that open and close a file respectively. FileOpen returns a reference number if the file opened ok (0 otherwise). Pass this reference number to FilePeek along with the number of the sector you wish to dump.

Figure 1. is a sample card that I use to dump the contents of the file. The dump is presented in a mono-spaced font to keep it "clean-looking". The forward and back arrows allow the user to "browse" sectors by moving 1 sector forward or backward. The scripts for these two buttons is fairly obvious so I've left them to the reader. The scripts for open, close are contained in listing 4.

- file open button

```
on mouseUp
    global fileName, DirectoryID
    global fileRefNum, theSector

    put getFileNameToOpen() into it
    put item 1 of it into fileName
    put item 2 of it into DirectoryID
    put fileName into card field "file name"

    if fileName is not empty then
        Put FileOpen( fileName, DirectoryID ) into fileRefNum
    end if
```

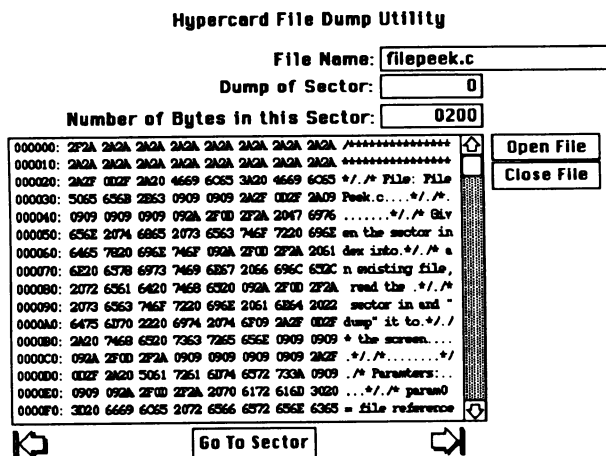



Figure 1

```
end mouseUp
```

- Go to Sector

```
on mouseUp
```

```
global fileRefNum, theSector
```

```
if filerefnum is empty then
    answer "You have to open a file first, silly!"
else
```

```
    ask "Read what sector?" with "0"
```

```
    if (it is not empty) then
        put it into theSector
        Peekat theSector
```

```
    end if
end if
```

```
end mouseUp
```

- file close

```
on mouseUp
```

```
global fileRefNum
```

```
put FileClose( fileRefNum ) into it
- ignore the result of a closed file
end mouseUp
```

Listing 4. Scripts for the buttons

An interesting variation on this theme would be to dump the files in reverse order so that it prints in ascending order on the LaserWriter. Several easy solutions exist for this problem, and I'll leave it as this month's exercise to the reader. As a hint, I can think of two ways to do this off hand, one easy but inefficient, the second a little more challenging: (1) create a card for every sector (make sure your file is small first!) or (2) Find out how many sectors the file contains before starting the dump (a wonderful opportunity to try your hand at XFCN writing).

Note on a bug in last month's article:

Last month's XFCN, GetFileName, incorrectly returns the working directory id of the file selected from SFGetFile. Change the call to GetFileNameToOpen to Read:

```
if( GetFileNameToOpen(types,numTypes,FileName,&FileWDID ) )(
    temp = (long)FileWDID & 0xFFFF;
    NumToStr( paramPtr, temp, &WDIDString );
    PtoCstr( WDIDString );
```

Listing 1: FilePeek.c

```
/******
/* File: FilePeek.c */
/*
/*
/* Given the sector index into */
/* an existing file, read the*/
/* sector in and "dump" it to*/
/* the screen */
/*
/*
/* Paramters: */
/* param0 = file reference num */
/* ( file is open ) */
/* param1 = sector number */
/* ( 1 sector = 512 bytes)*/
/*
/* To Build: */
/*
/* (1) Create a project using*/
/* this file as well as the */
/* XCMD.Glue.c file. (Set */
/* project type to XCMD (or */
/* XFCN) from the Project menu.*/
/*
/* (2) Bring the project up to */
/* date.
/*
/* (3) Build Code Resource. */
/*
/* (4) Use ResEdit to copy the */
/* resource to your stack. */
/******
```

```
#include <MacTypes.h>
#include <OSUtil.h>
#include <MemoryMgr.h>
#include <FileMgr.h>
#include <ResourceMgr.h>
#include <pascal.h>
#include <strings.h>
#include "HyperXCMD.h"
#include "HyperUtils.h"
```

```
#define SECTOR 512 /**size of input buffer**/
#define NUMROWS 8 /**across each line**/
#define NUMLINES 32 /**lines of data**/
#define NUMBYTES 16 /**bytes per row of data**/
#define SPACE 0x020 /**the space character**/
```

```
long paramtoNum();
void appendChar();
void CopyStrToHandle();
char *CopyAscii();
```

```
pascal void main( paramPtr )
    XCmdBlockPtr paramPtr;
{
    short err;
    short fref;
    short lc; /* line count */
    short rc; /* row count */
    long cnt;
    long blk; /* sector number */
    Handle outData;
    short *rPtr; /* per row */
    char *cPtr; /* for ASCII */
    char *aPtr; /* points to ascii */
```

```

char *buf;
char  asciiStr[32];
char  curntLine[256];
Str31  numString;

outData = 0L;
if( paramPtr->paramCount == 2 ){
    /** expect two parameters ***/
    /** (1) Get our input parameters ***/
    fref = (short)paramtoNum( paramPtr, 0 );
    blk  = paramtoNum( paramPtr, 1 );

    /** (2) Read a buffer of data ***/
    blk = blk * SECTOR;
    err = SetFPos( fref, fsFromStart, blk );
    if ( !err ){
        cnt = SECTOR;

        /** We need to keep this data locked ***/
        /** Since we can't trust callbacks not ***/
        /** to move stuff, we allocate the ***/
        /** buffer as non-relocatable. ***/
        buf = NewPtr( cnt );
        err = FSRead( fref, &cnt, buf );

        if ( err != noErr && err != eofErr ){
            paramPtr->returnValue = 0L;
            return;
        }
    }

    /** result is returned to hypercard as ***/
    /** a null terminated string ***/
    outData = NewHandle( 0L );

    /** (3) Start filling the output buffer***/
    *curntLine = '\0';

    /** first the number of bytes read in ***/
    NumToHex( paramPtr, cnt, 4, &numString );

    appendChar( PtoCstr( (char *)&numString ), '\r' );
    CopyStrToHandle( &numString, outData );

    /** point to the input data ***/
    rPtr = (short *)buf;

    for( lc = 0; lc < NUMLINES; ++lc ){
        *curntLine = '\0';

        /** blk is the sector address ***/
        NumToHex( paramPtr, blk, 6, &numString );
        PtoCstr( (char *)&numString );
        strcat( curntLine, &numString );
        appendChar( curntLine, ':' );
        appendChar( curntLine, SPACE );
        aPtr = asciiStr;
        blk += NUMBYTES;

        for( rc = 0; rc < NUMROWS; ++rc ){

            /** convert eight shorts per row ***/
            /** if we overrun the buffer, draw ***/
            /** whatever data follows it... ***/
            NumToHex( paramPtr, (long)rPtr, 4, &numString );
            strcat( curntLine, PtoCstr( (char *)&numString );
            appendChar( curntLine, ' ' );

            cPtr = (char *)rPtr;
            aPtr = CopyAscii( aPtr, *cPtr++ );
            aPtr = CopyAscii( aPtr, *cPtr++ );
            rPtr++;
        }
    }
}

```

```

    *aPtr = '\0'; /** terminate the ascii data ***/
    strcat( curntLine, asciiStr );
    appendChar( curntLine, '\r' );

    /** move line into output buffer ***/
    CopyStrToHandle( curntLine, outData );
}

DisposPtr( buf );
}

cnt = GetHandleSize( outData );
*(outData + cnt) = '\0';
paramPtr->returnValue = outData;
}

long paramtoNum( paramPtr, i )
XCmdBlockPtr paramPtr;
short i;
/*****
 * Given a handle to an input
 * argument in the paramBlk
 * return an integer representation
 * of the data.
 *****/
{
    Str31 theStr;

    HLock( paramPtr->params[ i ] );
    ZeroToPos( paramPtr, *(paramPtr->params[ i ]), &theStr );
    HUnlock( paramPtr->params[ i ] );
    return( StrToLong( paramPtr, &theStr ) );
}

void appendChar( theStr, theChar )
char *theStr;
char theChar;
/*****
 * append the character passed
 * to the end of the string
 *****/
{
    long len = strlen( theStr );
    char *theEnd;

    theEnd = theStr + len;
    *theEnd++ = theChar;
    *theEnd = '\0';
}

char *CopyAscii( outStr, theChar )
char *outStr;
char theChar;
/*****
 * if the character passed
 * in the input stream is a
 * printing character, append
 * it to the output string,
 * otherwise, append the '.'
 *
 * return the update output
 * string.
 *****/
{
    if ( theChar >= SPACE && theChar <= 0x0D8 )
        *outStr++ = theChar;
    else
        *outStr++ = '.';

    return( outStr );
}

void CopyStrToHandle( theStr, hand )

```

```

    char *theStr;
    Handle hand;
    /*****
    * Copy the input data to the
    * output handle.
    *
    *****/
    {
        long cnt = strlen( theStr );
        long oldSize = GetHandleSize( hand );

        SetHandleSize( hand, oldSize + cnt );
        BlockMove( theStr, *hand + oldSize, cnt );
    }

```

Listing 2: FileOpen.c

```

    /*****
    * File: FileOpen.c
    *
    * open the file whose name and
    * working directory id are
    * passed as parameters. This
    * information can be obtained
    * using GetFileNameToLoad...
    *
    * Parameters:
    * param0 = file name num
    * param1 = directory id
    *
    * Out:
    * File RefNum if opened, 0
    * otherwise
    *
    *****/

#include <MacTypes.h>
#include <OSUtil.h>
#include <MemoryMgr.h>
#include <FileMgr.h>
#include <ResourceMgr.h>
#include <pascal.h>
#include <strings.h>
#include "HyperXCmd.h"
#include "HyperUtils.h"

pascal void main( paramPtr )
    XCmdBlockPtr paramPtr;
{
    char *filename;
    Str31 str, fName;
    short wdid, refnum;

    HLock( paramPtr->params[0] );
    ZeroToPas( paramPtr, *(paramPtr->params[0]), &fName );
    HUnlock( paramPtr->params[0] );

```

```

    /* convert the wdid to a usable form */
    HLock( paramPtr->params[1] );
    ZeroToPas( paramPtr, *(paramPtr->params[1]), &str );
    HUnlock( paramPtr->params[1] );
    wdid = (short)StrToNum( paramPtr, &str );

    if( FSOpen( &fName, wdid, &refnum ) == noErr )
        NumToStr( paramPtr, (long)refnum, &str );
    else
        NumToStr( paramPtr, 0L, &str );

    paramPtr->returnValue = PasToZero( paramPtr, &str );
}

```

Listing 3: FileClose.c

```

    /*****
    * File: FileClose.c
    *
    * XCMD to access the file mgr
    * call FSClose
    * Parameters:
    * param0 = file reference from
    * the fileopen xcmd
    *
    *****/

#include <MacTypes.h>
#include <OSUtil.h>
#include <MemoryMgr.h>
#include <FileMgr.h>
#include <ResourceMgr.h>
#include <pascal.h>
#include <strings.h>
#include "HyperXCmd.h"
#include "HyperUtils.h"

pascal void main( paramPtr )
    XCmdBlockPtr paramPtr;
{
    short refnum;
    char str[256];

    /* convert the refnum to a usable form */
    HLock( paramPtr->params[0] );
    ZeroToPas( paramPtr, *(paramPtr->params[0]), &str );
    HUnlock( paramPtr->params[0] );
    refnum = (short)StrToNum( paramPtr, &str );

    /* Normally we should check the result*/
    /* code but it won't kill us to ignore*/
    /* the result of a close file call */
    if ( FSClose( refnum ) )
        ;
    paramPtr->returnValue = 0L;
}

```



HyperChat™

XCMD Corner: HFS Tree Climbing



HyperCard

Donald Koscheka
Arthur Young & Co.
MacTutor Contributing Editor
Volume 5, Number 8

Hyper Lite

In keeping with the spirit of the summer season, this month's XCMD is light and easy. Not too long ago, I promised to provide an XCMD that, given the name and directory id of a file, returned that file's full pathname. Such a utility is eminently useful in the current incantation of Hypercard.

Hypercard is fully capable of opening a file, reading and writing its contents and then closing the file. Unfortunately, unless you know the full path name of the file, you are limited to accessing files in Hypercard's working directory.

One of the first published XCMDs was a little gem from Steve Maller of Apple Computer Inc. that eliminates this MSDOS-compatible feature of Hypercard. Steve's FileName.p presents the user with the standard file package and returns the full pathname of the file specified.

If you are a regular reader of this column, you'll recall that I translated Steve's XCMD from pascal to "C". You also know that my version lost something in the translation. Rather than return a full pathname, GetFileName.c returns the file's name and working directory id. I felt this was more useful because the file's name and wdid open up access to most of the file manager calls for the XCMD programmer. Still, if you want to continue to use Hypercard's built-in file I/O, getFileName won't be of much use since it doesn't return the full pathname required by hypercard.

Listing 1 (FullPathName.c) is a simple XCMD that accepts the filename and working directory id as input and returns the fullpathname of the file as output.

As is often the case, this XCMD first converts the input data from c-string format to pascal strings and numbers as needed. Note that the default return value will be the name of the file that was passed in param[0].

A Review of HFS

As a rule, we don't come into contact with full pathnames very often on the Macintosh so the format may require a little explaining. If you want to explore the Hierarchical File System in greater detail, take the time to read Chapter 19 of volume IV of Inside Macintosh. If you don't have volume IV handy, the following discussion will get help you understand the code.

Devices that can store and retrieve data on the Macintosh are called "volumes". Files are organized on volumes in folders. Folders can contain other folders yielding a hierarchical or "Tree structure". The order in which folders are stored in this tree structure specify a path to any file that is contained in that folder.

The root of this tree is the volume name. If we stick a ":" onto the end of the volume name, we have the beginning of a full pathname. In other words full pathnames always start with the

volume name. Each path in the pathname is delineated by the ":". Thus, Hd:folder:file specifies a valid pathname. The last item in the pathname is the name of the file itself. The folder that holds this file is referred to by a unique directory id. When the user selects a file using SFGetFile or SFPutFile, the file manager returns the file's name as well as the directory id. For historical purposes, the directory id is stored in the vRefNum field of the reply record. This little tweak is how Apple was able to make HFS backward compatible with the flat-file system.

Tree Climbing

The folder that contains the file represents the deepest node on the tree. The next folder up the hierarchy is called the parent. Knowing the parent's id, we can call PBGetCatInfo to get its name as well as the directory id of its parent folder. We climb the tree by continuously feeding a parent directory id into PBGetCatInfo until we detect an error (no more folders on this path).

Tree climbing suggests recursion. The routine ClimbTree in Listing 1 calls PBGetCatInfo to get information about the directory whose id was passed in. We kick it off by first getting information about the folder that holds the file. This is the directory whose id was returned in the reply record.

Each activation of climbTree uses the same catalog parameter block (cpb) which was allocated in the stack frame of FullPathName. This is legal because the only information that is unique to each activation is the directory id (passed as a parameter in child) as well as a pointer to the string the PBGetCatInfo uses to store the folder's name. Each activation allocates memory for the name string and sets cpb's ioNamePtr field to point to its own copy of the string.

I allocate the string, folder_name, in the heap and point to it on the stack because successful recursion dictates that one be mindful of the stack at all times. Even if there is no danger of overflowing the stack, a little defensive programming goes a long way.

A nice feature of the recursion is that we can build the string in forward order. Although we walked the tree backwards, we don't actually put the full pathname together until we reach the root which is detected by discovering that it has no parent folder. Once we reach the root, the stack unwinds in the order of root->parent->child->filename.

The following script, working with GetFileName.c (See MacTutor Vol. 5 No. 6) returns the full pathname of a file selected via standard dialog:

```
on mouseup
    put getfilenametoOpen() into it
    if item 1 of it is not empty then
        put Fullpathname(item 1 of it, item 2 of it) into it
```

```

end if
end mouseup

```

Because Hierarchical File System can't handle pathnames longer than 256 characters, I limit the string returned by Fullpathname to 256 characters. If you want to return the full path regardless of length, you can modify the concat routine in "HyperUtils.c" to "grow" the output string on demand.

Listing: FullPathName.c

```

/*****
 * File: FullPathName.c
 * Given the name and working*
 * directory of a file, walk the*
 * directory tree to determine *
 * what the full pathname of the*
 * directory is... */
 * Parameters:
 * param0 = file name num */
 * param1 = directory id */
 * Out:
 * full pathname of the working*/
 * directory
 * Once again, I am indepted to*/
 * Steve Moller of Apple */
 * Computer Inc for illuminating*/
 * this oft too dark realm of*/
 * the toolbox. */
 *****/
#include <MacTypes.h>
#include <OSUtil.h>
#include <MemoryMgr.h>
#include <FileMgr.h>
#include <ResourceMgr.h>
#include <pascal.h>
#include <strings.h>
#include <hfs.h>
#include "HyperXCmd.h"
#include "HyperUtils.h"
#define nil 0L

char colon[2] = "\p";

void ClimbTree( child, cpb, fullName )
long child;
CInfoPBPtr cpb;
char *fullName;
/*****
 * Climb the directory tree
 * until we reach the root
 * Allocate the records in the
 * heap to keep the stack frame
 * as small as necessary. Too
 * large a stack frame can
 * lead to a case of terminal
 * terminal.
 * child is the working directory
 * id of the "current folder", vol
 * is the volume reference number and
 * fullName points to the
 * output string
 *****/
{
StringPtr folder_name = (StringPtr)NewPtr( 256 );
/**** setting the file directory index to -1****/
/**** lets us get information about the ****/
/**** directory whose id is specified in the****/
/**** ioDrDirID field of the parameter block****/
folder_name[0] = '\0';
cpb->dirInfo.ioNamePtr = (StringPtr)folder_name;
cpb->dirInfo.ioDrDirID = child;

```

```

if( PBGetCatInfo( cpb, 0 ) == noErr )
ClimbTree(cpb->dirInfo.ioDrParID, cpb, fullName);

Concat( (char *)fullName, (char *)folder_name );
Concat( (char *)fullName, (char *)&colon );
}
DisposPtr( folder_name);
)

```

```

pascal void main( paramPtr )
XCmdBlockPtr paramPtr;

```

```

{
Str31 str, fName;
short wdid;
WDPBRec theWDPB;
CInfoPBRec theCPB;
HPParamBlockRec theHPB;
char fullPath[256];
char part_Name[256]; /**** used in HPB ****/
char vol_Name[256];
OSErr err;

colon[0] = 1;
colon[1] = ':';

vol_Name[0] = '\0';
part_Name[0] = '\0';
/**** empty is the default answer ****/
paramPtr->returnValue = 0L;

HLock( paramPtr->params[0] );
ZeroToPas( paramPtr, *(paramPtr->params[0]), &fName );
HUnlock( paramPtr->params[0] );

/**** convert the wdid to a usable form ****/
HLock( paramPtr->params[1] );
ZeroToPas( paramPtr, *(paramPtr->params[1]), &str );
HUnlock( paramPtr->params[1] );
wdid = StrToNum( paramPtr, &str );

/**** First, we appeal to GetVInfo to get****/
/**** volume name that the file lives on ****/
part_Name[0] = '\0';
theHPB.volumeParam.ioNamePtr = (StringPtr)vol_Name;
theHPB.volumeParam.ioVRefNum = (short)wdid;

theHPB.volumeParam.ioVolIndex = 0;
if( PBHGetVInfo( &theHPB, 0 ) != noErr )
return;

/**** Next, use the working directory info ****/
/**** to walk the directory tree backwards ****/
/**** to the root directory ****/
theWDPB.ioNamePtr = (StringPtr)part_Name;
theWDPB.ioVRefNum = wdid;
theWDPB.ioWDProcID = 0;
theWDPB.ioWDIndex = 0;
if( PBGetWDInfo( &theWDPB, 0 ) != noErr )
return;

fullPath[0] = '\0';
theCPB.dirInfo.ioFDirIndex = -1;
theCPB.dirInfo.ioVRefNum = theHPB.volumeParam.ioVRefNum;

ClimbTree( theWDPB.ioWDDirID,
(CInfoPBPtr)&theCPB,
(StringPtr)fullPath );

/**** Climbing the tree yields the names of****/
/**** all the folders which we still need to ****/
/**** add the file's name to. ****/
Concat( (char *)fullPath, (char *)&fName );

```

```
paramPtr->returnValue = PasToZero( paramPtr, fullPath );
}
```

Listing: HyperUtils.H

```

/*****
 * HyperUtils.H
 * Header file for HyperUtils.c
 * routines...
 *****/
#define NIL 0L
#define UPFRONT -1L
void CenterWindow( WindowPtr wptr );
void Concat( char * str1, char * str2 );
void CopyPStr( char * pStr1, char * pStr2 );
short GetFileNameToOpen( SFTypelist typs, short typcnt, char
 *theName, short *theWDID );

```

Listing: HyperUtils.c

```

/*****
 * HyperUtils.c
 * A collection of useful
 * routines...
 *****/
#include <MacTypes.h>
#include <OSUtil.h>
#include <MemoryMgr.h>
#include <FileMgr.h>
#include <ResourceMgr.h>
#include <StdFilePkg.h>
#include "HyperXCmd.h"
#include "HyperUtils.h"

void CenterWindow( wptr )
    WindowPtr wptr;
/*****
 * Center a window in the current
 * screen port. Note: Does not
 * attempt to work with multi-screen
 * systems.
 * This code is inspired by a
 * similar routine written by Steve
 * Maller in MPW Pascal. Thanks Steve.
 *****/
{
    short hWindSize = wptr->portRect.right - wptr->portRect.left;
    short vWindSize = wptr->portRect.bottom - wptr->portRect.top;
    short hSize = wptr->portBits.bounds.right - wptr-
    >portBits.bounds.left;
    short vSize = wptr->portBits.bounds.bottom - wptr-
    >portBits.bounds.top;

    MoveWindow( wptr, ( hSize - hWindSize ) / 2,
                ( vSize - vWindSize + 20 ) / 2, false );
}

void Concat( str1, str2 )
    char *str1;
    char *str2;
/*****
 * Append string 2 to the end of
 * string 1. Both strings are
 * pascal-format strings.
 * str1 must be large enough to hold
 * the new string and is assumed to
 * be of Type Str255 (a pascal string)
 *****/
{
    short len1 = *str1; /***number of chars in string 1***/
    short len2 = *str2++; /*** number of chars in string 2***/
    char *temp; /*** string pointer ***/

    if( len1 + len2 > 255 )
        len2 = 255 - len1;
    *str1++ += len2 ; /***add sizes together to get new size***/

```

```

    temp = str1 + len1; /*** move to end of string 1***/
    while( len2 ){
        *temp++ = *str2++; /*** add char to temp and move along***/
        len2--; /*** until all characters are added***/
    }
}

```

```

void CopyPStr( pStr1, pStr2 )
    char *pStr1;
    char *pStr2;
/*****
 * Copy the contents of pstr1 into
 * pstr2. The strings are assumed
 * to be of type STR255 (length byte
 * precedes data
 *****/
{
    short i;
    char *tstr;

    tstr = pStr2;
    for( i = 0; i <= *pStr1; i++ )
        *tstr++ = *pStr1++;
}

```

```

short GetFileNameToOpen( typs, typCnt, theName, theWDID )
    SFTypelist typs; short typCnt;
    char *theName; short *theWDID;
/*****
 * Invokes SFOpenFile to query the
 * user for the name of a file to open.
 * In: List of types of files to
 * filter for (up to 4)
 * Out: fileName if picked in theName
 * working directory in theWDID
 * nil otherwise
 * the file's volum ref num.
 * ( Note that the space for the
 * string must be allocated by the
 * caller).
 *****/
{
    Point where;
    char prompt[1];
    SFReply reply;
    GrafPort *oldPort;
    WindowPtr dlogID;
    prompt[0] = '\0';
    /*** Get and put up the standard file ***/
    /*** dialog. You will only see the file***/
    /*** types that you filtered for. If ***/
    /*** you filtered for no files, then ***/
    /*** all files will display ***/
    GetPort( &oldPort );
    dlogID = GetNewDialog( (short)getIdID, (Ptr)NIL,
    (Ptr)UPFRONT );
    SetPort( dlogID );
    CenterWindow( dlogID );
    where.h = dlogID->portRect.left;
    where.v = dlogID->portRect.top;
    LocalToGlobal( &where );
    SFGGetFile( where, prompt, (Ptr)NIL, typCnt, typs, (Ptr)NIL,
    &reply );
    DisposDialog( dlogID );
    SetPort( oldPort );
    /*** If the user selected a file, let's ***/
    /*** get the information about it ***/
    if (reply.good){
        *theWDID = reply.vRefNum;
        PtoCstr( (char *)&reply.fName );
        strcpy( theName, &reply.fName );
    }
    return( reply.good );
}

```



HyperChat™

The Blessed Folder



Donald Koscheka
Arthur Young & Co.
MacTutor Contributing Editor
Volume 5, Number 9

A colleague related an interesting experience that occurred to him at MacHacks this year. He asked an Apple Engineer whether he could mount Appleshare volumes under program control. The engineer asked him why he'd want to and went on to explain that it's simply not something you do. If you need to mount Appleshare volumes, use the chooser.

This struck me as odd because I've always believed that your programs should be able to do anything that the user can do.

Imagine some inexperienced user being told that "the server has unexpectedly shut down". Consider the options that this error message presents to the uninitiated: Ask someone what's going on, call Apple's customer service department or read the manual.

Each one of these options punishes the user for a mistake that he or she did not make! Why not have a little task running somewhere that tries to reconnect the user and, failing that, alerts the poor soul that they may have to visit the chooser to regain access to the server?

Perhaps too many people at Apple are falling into that old trap of believing in everything they read. The human interface guidelines should be just that, guidelines. When you start taking guidelines as gospel, you start closing the door on creativity and intuition.

It seems to me that the good people at Apple need to spend less time saying "You can't do that" and spend more time asking, "Why can't you do that?"

The file manager is another area where the Macintosh tends to dump problems in the user's lap. Consider the "Where is ..." dialog that pops up in Hypercard from time to time. Imagine how intimidating that must be to the neophyte, "If the computer can't find it, I'm sure not going to have any luck".

A better user interface might suggest to the user that the Mac can search the disk on the user's behalf. It might take a while, but if the user doesn't have a clue as to where some file is, time really doesn't become an issue.

The problem with the file manager is that it relies too heavily on the user. Giving control to the user is a wonderful idea. Nonetheless, the user should be able to delegate tasks like searching back to the Macintosh.

Perhaps you need to store a preferences or help file somewhere. If the user moves this file on you, are you supposed to pop a dialog asking the user to find the file for you again? Of course not, you're going to store the file in a folder that you're guaranteed to have access to at all times. That folder is the blessed folder, so called because it contains the system file and startup application (typically the Finder). All you need is access to the blessed folder under program control and your problem is solved.

Fortunately, Apple does have a champion for developers who would like to unload some of the file management stuff from

the user. Jim Friedlander has published many useful tech notes for developer technical services describing how to do things like finding that blessed folder.

Listing 1 adds an interesting spin to this blessed folder business. This XCMD, called Volumes, returns a list of all mounted volumes. The volumes XCMD uses a handy little routine called pStrToField that adds a pascal format string to the end of a zero-terminated run of text. It's a useful way of adding names that you get from the toolbox which are almost always Pascal strings to a container that you can return to Hypercard.

If you're building a list, terminate each item with the 'r' (carriage return) character. If you are building items, then use a comma. If you don't want any delimiters, then pass a '0'.

Because pStrToField is a general purpose routine, you should add it to whatever library you use to store such routines. If you are a regular reader of this column, you know that I use a file called "HyperUtils.c" for such routines.

Retrieving the list of volumes is simple. Set some index counter to 1. Then repeatedly call PBHGetVInfo with the following fields set in the parameter block:

```
ioNamePtr    = pointer to a string to store the name in
ioVRefNum    = -1, tell file manager to use the volume index.
ioVolIndex   = your index counter.
```

Keep incrementing ioVolIndex until PBHGetVInfo returns an error. That's a good indication that no more volumes were found.

Each time that pbHGetVInfo succeeds, the name of the found volume will be returned in ioNamePtr as a Pascal string. We tack the obligatory colon to the end of this name and then add the new string to the volume list that we're building in vList.

After all is said and done, we set the last character in the container to 0 because Hypercard expects containers to be null terminated.

Knowing the names of all mounted volumes may not seem useful at first blush but given a little time, I'm sure you'll find many interesting ways to use this XCMD.

Once we know all the volume names on line, we can find all the blessed folders. A blessed folder is one that contains the system and startup application (usually the finder). You can find the blessed folder by inspection; it's the one whose folder contains a very small image of a Macintosh.

Knowing the blessed folder does have applicability in very many cases. Perhaps your XCMD expects to store a preferences file in the blessed folder. At any rate, knowing how to find the blessed folder should be a part of every Mac programmer's repertoire. The XCMD in listing 2, BlessedFolder, does just that. It's based on tech note #129 by Jim Friedlander.

Note that this time we use PBHGetVinfo with the volume reference set to 0 and the index set to -1. By passing the name of a volume, we are telling the file manager to look up information about this volume using its name rather than a reference number or index.

The information returned by PBHGetVinfo gets a little fuzzy here. One of the fields returned is ioVFndrInfo which is an array of 8 long integers. The first entry in this array is the id of the blessed folder. I'm sure this array is documented somewhere in tech notes and if you want to further explore this information, that's a good place to start since IM volume IV seems to gloss over this field.

Once we know the directory id of the blessed folder, we can pass it to, climbtree (published last month), to reconstruct the full pathname of this folder. Climbtree works just as well for folders as it does for files. We start climbtree off with the directory id of the blessed folder which was assigned to theCPB.dirInfo.ioDrDirID. Climbtree also needs a volume reference number which we pass via ioVRefNum.

Climbtree requires that you declare a CInfoPBRec somewhere before calling it. This is because Climbtree is recursive, and I didn't want stack space being consumed by a declaration of the relatively large catalog info record with each activation. If you're a purist, you might take your chances with the heap and create a nonrelocatable block in each activation. If 8K seems like too much stack space to you, go ahead and declare theCPB as an automatic of climbtree.

This month's XCMDs fall into the category of "Now that I have them, what can I do with them". Play around for a while and see what you discover on your own. In the meantime, I'll be cooking up some useful applications for these XCMDs.

```

/*****
/* File: Volumes.c */
/* */
/* Return a list of all on-line*/
/* Takes no input and returns */
/* a carriage return delimited*/
/* list of all volumes currently */
/* on line */
/* */
/* _____ */
/* ©1989, Donald Koscheka */
/* All Rights Reserved */
*****/

#include <MacTypes.h>
#include <OSUtil.h>
#include <MemoryMgr.h>
#include <FileMgr.h>
#include <ResourceMgr.h>
#include <pascal.h>
#include <hfs.h>
#include <string.h>
#include "HyperXCmd.h"
#include "HyperUtils.h"

#define nil 0L

char colon[2] = "\p:";

short pStrToField( str, delim, list )
char *str;

```

```

char delim;
Handle list;
/*****
* Given a pascal string, append it to
* the end of the handle passed in list
* which is assumed to be a valid handle
* of length >= 0.
*
* delim is some character to stick on the
* end of the string to delimit it.
* if you want to build a list for presentation
* in a field, pass '\r' as the delimiter
*
* If you are building items pass a comma
*
* A value of 0 for delim is ignored. Pass
* 0 when you don't want a delimiter.
*****/
(
    long strlen; /* length of input string*/
    long oldHSize; /* size of input handle*/
    char *end; /* pointer to end of data */

    strlen = str[0]; /* length of string is in first byte*/
    oldHSize = GetHandleSize( list );

    SetHandleSize( list, oldHSize + strlen );
    end = *list + oldHSize;

    BlockMove( (char *)&str[1], end, strlen );

    if( delim ){
        oldHSize = GetHandleSize( list );
        SetHandleSize( list, oldHSize + 1 );
        end = *list + oldHSize;
        *end = delim;
    }
)

pascal void main( paramPtr )
XCmdBlockPtr paramPtr;
(
    short index;
    long len;
    char *end;
    HParamBlockRec theHPB;
    Handle vlist;
    char vol_Name[256];
    OSErr err;

    colon[0] = 1;
    colon[1] = ':';

    /*** empty is the default answer ***/
    vlist = NewHandle( 0L );

    /*** Search for every volume that is on-line***/
    index = 1;
    do( /*** Appeal to the volume manager***/
        /*** for the name of each volume ***/
        /*** that is known to it ***/

        vol_Name[0] = '\0';
        theHPB.volumeParam.ioNamePtr = (StringPtr)vol_Name;

        theHPB.volumeParam.ioVRefNum = (short)-1;

        theHPB.volumeParam.ioVolIndex= index;

        err = PBHGetVInfo( &theHPB, 0);

        if ( !err ){/** Add each Volume to List **/
            Concat( vol_Name, colon );
            pStrToField( (char *)&vol_Name, '\r', vlist );

```



```

        index += 1;
    }
} while (err == noErr);

/** once done, tack a 0 onto the end of vlist */
len = GetHandleSize( vlist );
SetHandleSize( vlist, len+1 );
end = *vlist + len;
*end = '\0';

paramPtr->returnValue = vlist;
}

```

Listing 1. Volumes.c

```

/*****
/* File: BlessedFolder.c */
/*
/* Given the name of a volume */
/* in params[0] */
/* returns the id of the */
/* blessed folder. */
/* ----- */
/* Based on tech note #129 */
/* by Jim Friedlander */
*****/

#include <MacTypes.h>
#include <OSUtil.h>
#include <MemoryMgr.h>
#include <FileMgr.h>
#include <ResourceMgr.h>
#include <pascal.h>
#include <hfs.h>
#include <string.h>
#include "HyperXCmd.h"
#include "HyperUtils.h"

```

```

#define nil 0L

```

```

char colon[2] = "\p:";
pascal void main( paramPtr )
    XCmdBlockPtr paramPtr;
{
    long sid; /** id of blessed folder */
    HParamBlockRec theHPB;
    CInfoPBRec theCPB; /** to reconstruct path name */
    char vName[256]; /** volume we're checking */
    char fullPath[256];
    OSErr err;

    HLock( paramPtr->params[0] );
    ZeroToPas( paramPtr, *(paramPtr->params[0]), &vName );
    HUnlock( paramPtr->params[0] );
    /** Given the name of a volume, */
    /** try to get the volume reference */
    /** number */
    theHPB.volumeParam.ioNamePtr = (StringPtr)vName;
    theHPB.volumeParam.ioVRefNum = (short)0;
    theHPB.volumeParam.ioVolIndex = -1;
    err = PBHGetVInfo( &theHPB, 0 );

    fullPath[0] = '\0';
    if ( !err ){
        theCPB.dirInfo.ioFDirIndex = -1;
        theCPB.dirInfo.ioDrDirID = theHPB.volumeParam.ioVFndrInfo[0];
        theCPB.dirInfo.ioVRefNum = theHPB.volumeParam.ioVRefNum;
        fullPath[0] = '\0';
        ClimbTree( theCPB.dirInfo.ioDrDirID,
            (CInfoPBPtr)&theCPB, (char *)fullPath );
    }
    paramPtr->returnValue = PasToZero( paramPtr, fullPath );
}

```

Listing 2. BlessedFolder.c



HyperChat™

XCMD Corner: CopyFile



Donald Koscheka
Arthur Young & Co.
MacTutor Contributing Editor
Volume 5, Number 10

CopyFile XCMD

From time to time I monitor the developer forums on AppleLink and MacNet to see what the rest of the Macintosh development community is up to. Recently, Richard Greenawalt of Foremost Computer Systems issued a request for some help. It seems he needed a routine that copies Macintosh files, data fork, resource fork and finder information.

Not one to miss an opportunity to write an interesting XCMD, I linked Rick back and told him that I would be happy to write the routine for him.

The moral of the story is: ask and you might receive. If you need some help with an XCMD or if this column just doesn't do it for you, let me know; I will be more than happy to help you out. Although I am up on AppleLink, I prefer to use MacNet (KOS-CHEKA). If you don't have access to a modem, write me care of MacTutor.

Copying a Macintosh file is not difficult as long as you keep in mind that Mac files are really two files in one. Each file on the Macintosh has a split personality - the data fork and the resource fork. Although the two forks act as one entity, the file manager treats them as separate files.

CopyFile is a handy little XCMD that lets you copy files from Hypercard. Actually the routine is written to work from any programming milieu which is why I've split it into two listings. The first consists of the Hypercard interface. Note that you need to pass both the file name and the working directory id. The second listing contains the actual code, void of any references to Hypercard. This approach lets me build libraries of usable routines that I can use with or without Hypercard, and I intend to use it for all future listings.

The CopyFile function expects to see the name of the input file, the name of the output file and the working directory id's of both. This gives you some flexibility in naming the copy as well as deciding what folder to put it in. For example, you can concatenate the name of the input file to "copy of " so that the duplicated file is called "copy of file". Put this concept in a loop and you get the finder like capability of creating files and naming them "copy of ...", "copy of copy of ..." and so on. I chose not to implement this approach because I think the user should have the opportunity to specify the name of the copied file.

Although there are several ways to go about copying a file, I chose a path that goes something like this: First, get the collective size of the data and resource fork along with the finder information about the input file. Using this information, attempt to create a file that is as large as the input file (ie as large as the data and resource fork combined). If the space can be allocated, then copy each fork in turn. If not, delete the newly created file and quit.

Space for each fork is allocated before we do the copy so that we can determine *a priori* whether the file will fit. To be safe, we reposition the file mark at the beginning of the file before we start copying. If, for any reason, the file copy fails, delete what remains of the file and return the error message to the caller.

The real work is done by the function, CopyFork. This routine will attempt to read the entire fork into a single buffer. Failing that, it divides the size in half until enough memory can be allocated to read some of the fork. Note that copyfork attempts to allocate a buffer large enough to read the entire fork into memory. If that much memory is not available, it keeps dividing the original size by two until a large enough buffer can be allocated.

That's file copying in a nutshell. It's not particularly difficult once you realize that you're really copying two files - the data fork and the resource fork. I've tried it with files up to 4 megabytes in size and it works fine.

```

/*****
/* File: FileCopy.c      */
/* param0 = file reference num */
/* ( file is open ) IN:      */
/*params[0] = name of input */
/*params[1] = wdid of input */
/*params[2] = name of output */
/*params[3] = wdid of output */
*****/
#include <MacTypes.h>
#include <OSUtil.h>
#include <MemoryMgr.h>
#include <FileMgr.h>
#include <ResourceMgr.h>
#include <pascal.h>
#include <string.h>
#include "HyperXCMD.h"
#include "HyperUtils.h"

pascal void main( paramPtr )
    XCmdBlockPtr paramPtr;
/*****
 * params[0] = name of input
 * params[1] = wdid of input
 * params[2] = name of output
 * params[3] = wdid of output
 *****/
( OSErr    err;
  short    inWD;
  short    outWD;
  long     temp;
  Str31    errCode;
  char     inFile[256];
  char     outFile[256];
  paramPtr->returnValue = 0L;
  /*** (1) Get input parameters ***/
  HLock( paramPtr->params[0] );
  ZeroToPas( paramPtr, *(paramPtr->params[0]), &inFile );
  HUnlock( paramPtr->params[0] );
  inWD = (short)paramtoNum( paramPtr, 1 );

```

```

HLock( paramPtr->params[2] );
ZeroToPas( paramPtr, *(paramPtr->params[2]), &outFile );
HUnlock( paramPtr->params[2] );
outWD = (short)paramtoNum( paramPtr, 3 );
temp = (long)CopyFile( inFile, inWD, outFile, outWD );
/** Flush the output volume */
err = FlushVol( 0L, 0 );
NumToStr( paramPtr, temp, &errCode );
paramPtr->returnValue = PasToZero( paramPtr, &errCode );
}

```

LISTING 1: CopyFile XCMD.

```

OSErr CopyFork( inref, outref, siz )
short inref; short outref; long siz;
/*****
* Given that the caller has opened
* a fork and passed you the names of
* the input, copy the number of bytes
* from the input fork to the output
* fork.
* The input mark should be set to
* start of fork.
* We use a "semi-smart" algorithm
* to do the copy. If the entire
* fork can be copied, we try doing
* that, otherwise, we keep dividing
* the size by two until we get enough
* room to read some data in.
*****/
{
    OSErr rd_err = noErr;
    OSErr wrt_err = noErr;
    long rd_len; /** actual bytes read */
    Ptr inbuf;
    /** make sure that the size is even */
    if( siz % 2 )
        ++siz;
    if( siz > 0 ){
        do{
            inbuf = NewPtr( siz );
            if( !inbuf )
                siz = ( siz >> 1 );
        }while( !inbuf );
        /** inbuf is the buffer that we */
        /** read the data into. If not*/
        /** allocated, don't attempt to do */
        /** the read */
        if( inbuf ){
            do{
                rd_len = siz;
                rd_err = FSRead( inref, &rd_len, inbuf );
                wrt_err = FSWrite( outref, &rd_len, inbuf );
            }while( !rd_err && !wrt_err );
            DisposPtr( inbuf );
        }
    }
    return( wrt_err );
}

OSErr CopyFile( inFile, inWD, outFile, outWD )
char *inFile;
short inWD; char *outFile;
short outWD;
/*****
* (1) Determine the size of the input
* file.
* (2) Attempt to allocate that
* much space for the output file.
* (3) If allocation successful,
* create the output file.
* (4) Once the file is created,
* copy the data fork, the resource
* fork and the finder information

```

```

* from the input file.
* The file will be called "copy of..."
* Each time we create the file, first
* see if that name exists, if so, keep
* sticking "copy of" onto the name.
*****/
{
    OSErr err;
    OSErr err2;
    short inref;
    short outref;
    long data_eof = 0L;
    long rsrc_eof = 0L;
    FInfo fndrinfo;
    /** (2) Determine how big the input file is */
    if( (err = FSOpen( inFile, inWD, &inref )) == noErr ){
        err = GetEOF( inref, &data_eof );
        err = FSClose( inref );
    }
    if( (err = OpenRF( inFile, inWD, &inref )) == noErr ){
        err = GetEOF( inref, &rsrc_eof );
        err = FSClose( inref );
    }
    /** (2) Create output file and allocate space */
    if( (err = GetFInfo( inFile, inWD, &fndrinfo )) != noErr )
        return( err );
    if( (err = Create( outFile, inWD, fndrinfo.fdcCreator,
        fndrinfo.fdtype )) != noErr )
        return( err );
    /** (3) Try to allocate enough space for both */
    /** forks. Note that if we get enough space. */
    if( (err = FSOpen( outFile, outWD, &outref )) != noErr )
        return( err );
    if( (err = SetEOF( outref, data_eof )) != noErr ){
        err2 = FSClose( outref );
        err2 = FSDelete( outFile, outWD );
        return( err );
    }
    err2 = FSClose( outref );
    err = OpenRF( outFile, outWD, &outref );
    if( (err = SetEOF( outref, rsrc_eof )) != noErr ){
        err2 = FSClose( outref );
        err2 = FSDelete( outFile, outWD );
        return( err );
    }
    err2 = FSClose( outref );
    /** (4) Copy the Data fork */
    err = FSOpen( inFile, inWD, &inref );
    if( !err ){
        err2 = SetFPos( inref, fsFromStart, 0L );
        err = CopyFork( inref, outref, data_eof );
        err2 = FSClose( inref );
        err2 = FSClose( outref );
    }
    if( err ){
        err2 = FSDelete( outFile, outWD );
        return( err );
    }
    /** (5) Now copy the resource fork */
    err = OpenRF( inFile, inWD, &inref );
    if( !err ){
        err2 = SetFPos( inref, fsFromStart, 0L );
        err = OpenRF( outFile, outWD, &outref );
        err = CopyFork( inref, outref, rsrc_eof );
        err2 = FSClose( inref );
        err2 = FSClose( outref );
    }
    if( err ){
        err2 = FSDelete( outFile, outWD );
        return( err );
    }
    return( noErr );
}

```

LISTING 2: CopyFile and CopyFork Functions.



HyperChat™

XCMD Corner: Catalog XCMD



HyperCard

Donald Koscheka
Arthur Young & Co.
MacTutor Contributing Editor
Volume 5, Number 11

Bullet Proofing

Recently, Joe Palooka wrote to complain about a bug in the GetFileName XCMD that appeared in this column a few months back. Apparently, the xcmd bombed if you passed it more than four file types. But this letter raises a more important issue: how much bulletproofing should I do to my code examples?

Before you jump to the obvious conclusion, allow me to make my case. Bulletproofing my code examples carries three liabilities for this column, size and clarity and time. The former is obvious. This magazine can carry only a certain number of pages per issue. Exceeding that page limit on a regular basis will result in my column getting bumped or serialized. I would like to avoid either case.

Clarity is not so obvious and, in fact, is a function of size. The larger a program is, the more unreadable it becomes. Bulletproofing code tends to be fairly routine, and one can almost apply a formula to XCMDs for bulletproofing: (1) Are the input parameters defined? (2) if so, are they properly conditioned (e.g. correct number of data).

The final liability is time. My years of experience have taught me that a large amount of a programming project's time is spent anticipating exceptions. Working under a monthly deadline, it is often necessary to eliminate this step. I don't see any contradiction here. If you intend to use my code in your own product, then it's incumbent on you test its effectiveness. Bulletproofing my examples will only make your job harder and even then, I have no way of anticipating your exceptions (i.e. what happens to subroutine x if I make this small change to its parameters?)

This code tends to take the form:

```
if( paramPtr->params[i] && **(paramPtr->params[i]) )...
```

Testing the input parameters is something else entirely and can easily exceed the size of the "functional code".

This is not to belittle the need for bulletproof code. Far from it. If you intend to use the code in this column, you should use it as a starting point. In general, if you intend to use somebody else's code, you should seek answers to these questions: what does the code do? what doesn't the code do?

At any rate, I will leave this issue on your hands. Write to the editor and let him know your feelings.

Catalog

One good XCMD deserves another. Recently, Joe Zuffoletto called me to ask whether I had any intentions of writing an XCMD that returns a catalog of a given directory. You may recall that Joe gave us the series on adding standard Mac

windows to Hypercard. Such code cannot go unrewarded and so this month's xcmd ...

Catalog accepts parameters in one of two forms. If you pass a pathname in the first parameter, it will return a catalog of all files in that directory. If the first parameter is empty, then Catalog will use the second parameter as directory reference number. This second parameter requires further explaining by way of the file manager.

When the user selects a file from the standard file dialog, the name as well as the volume reference number is passed back to the caller. Under HFS, the volume reference number masquerades as a working directory id. In reality, this reference number acts as either a volume reference number or a working directory id. How the file manager tells them apart is subtle but important:

Volume reference numbers are small negative integers such as -1, -2, -3... In fact, if you pass such a number to Catalog, you will get a listing of the root directory of the volume in question.

Working directory id's are x. Working directories must not be confused with directory id's. Each file has a unique id assigned to it called a directory id. This id is a long integer that uniquely identifies a file or folder. Working directory id's by contrast are integers and are not unique for a given folder. Working directories are very much like file reference numbers. When a working directory is opened by the call, PBOpenWD, a reference number is returned in the vRefNum field.

Listing 1:

```
/******  
/* File: Catalog.c */  
/*  
/* Given the reference number of*/  
/* a working directory or volume*/  
/* return a list of all files & */  
/* folders in that directory */  
/*  
/* if a name is given, use it, */  
/* otherwise, use the id passed*/  
/******
```

```
#include <MacTypes.h>  
#include <OSUtil.h>  
#include <MemoryMgr.h>  
#include <FileMgr.h>  
#include <ResourceMgr.h>  
#include <pascal.h>  
#include <string.h>  
#include <hfs.h>  
#include "HyperXCmd.h"  
#include "HyperUtils.h"
```

```
#define nil 0L
```

```
extern GetCatalog();
```

```

pascal void main( paramPtr )
XCmdbContext paramPtr;
{
    Handle    catalog;
    Str31     str;
    short     dirID;

    /** initialize the output container    */
    catalog = NewHandle( 0L );

    /** convert the wdid to a usable form */
    if( paramPtr->params[1] ){
        HLock( paramPtr->params[1] );
        ZeroToPas( paramPtr, *(paramPtr->params[1]), &str );
        HUnlock( paramPtr->params[1] );
        dirID = (short)StrToNum( paramPtr, &str );

        /** given the id of a directory,    */
        /** return a catalog for that directory */
        GetCatalog( dirID, catalog );
    }
    else if( paramPtr->params[0] ){
        char path[256];
        CInfoPbRec catPB;
        WDPBRec     wdPB;

        HLock( paramPtr->params[0] );
        ZeroToPas( paramPtr, *(paramPtr->params[0]), &path );
        HUnlock( paramPtr->params[0] );

        /** get a directory id to this path */

        catPB.dirInfo.ioNamePtr = (StringPtr)path;
        catPB.dirInfo.ioFDirIndex = 0;
        catPB.dirInfo.ioVRefNum = 0;
        if( PBGetCatInfo( &catPB, 0 ) == noErr )
            if( catPB.dirInfo.ioFAttr & 0x010 ) {
                it's a directory */
                wdPB.ioNamePtr = (StringPtr)path;
                wdPB.ioWDProcID = 0;

                if( PBOpenWD( &wdPB, 0 ) == noErr ){
                    wdPB.ioWDVRefNum = catPB.dirInfo.ioVRefNum;
                    wdPB.ioWDDirID = catPB.dirInfo.ioDrDirID;
                    GetCatalog( wdPB.ioVRefNum, catalog );
                    PBCloseWD( &wdPB, 0 );
                }
            }

        /** append a null to the end of the    */
        /** the directory for Hypercard    */
        AppendCharToHandle( catalog, '\0' );

        paramPtr->returnValue = catalog;
    }
}

```

Listing 2:

```

#include <MacTypes.h>
#include <OSUtil.h>
#include <MemoryMgr.h>
#include <FileMgr.h>
#include <ResourceMgr.h>
#include <pascal.h>
#include <hfs.h>
#include <string.h>
#include "HyperXCmdbContext.h"
#include "HyperUtils.h"

```

```

/** If you read this column */
/** column on a regular basis */
/** you may want to add these */
/** routines to HyperUtils.c */
/** If you don't have Hyper-

```

```

/** utils.c, you can obtain it */
/** from MacTutor for a small */
/** handling charge    */

```

```
#define SYNC 0
```

```

AppendCharToHandle( theHand, theChar )
    Handle theHand;
    char theChar;

```

```

/*****
 * Given a valid handle, append
 * the character passed in to
 * the end of the handle
 */

```

```

 * This is a useful way to embed
 * \r, \t or \0 into a container
 * for use by hypercard.
 *****/

```

```

{
    long hsz = GetHandleSize( theHand );
    char *dirP;

```

```

    SetHandleSize( theHand, hsz + 1 );
    dirP = *theHand + hsz;
    *dirP = theChar;
}

```

```

GetCatalog( wdref, dirH )
    short wdref;
    Handle dirH;

```

```

/*****
 * Get a listing of the directory
 * passed in.
 * In:

```

```

 * wdref == reference number of the
 * the desired working directory
 * dirH == handle to the output container.
 * Note that we allocate all structures
 * in the heap to minimize the impact
 * a high directory valence might have
 * on the stack.
 *****/

```

```

{
    OSErr done = 0; /* goes true when done searching */
    CInfoPbPtr catPB = (CInfoPbPtr)NewPtr( sizeof(CInfoPbRec));
    WDPBPtr     wdPB = (WDPBPtr)NewPtr( sizeof(WDPBRec));
    char *fname = (char *)NewPtr( 256L );

```

```

    if( catPB && wdPB && fname ){
        catPB->dirInfo.ioNamePtr = (StringPtr)fname;
        catPB->dirInfo.ioFDirIndex = 0;
        catPB->dirInfo.ioVRefNum = wdref;

```

```

        do{
            *(catPB->dirInfo.ioNamePtr) = '\0';
            catPB->dirInfo.ioFDirIndex++;
            catPB->dirInfo.ioDrDirID = 0;

```

```

            if( done = PBGetCatInfo( catPB, SYNC ) ) == noErr {
                pStrToField( fname, ' ', dirH );
                if( catPB->dirInfo.ioFAttr & 0x010 ) {
                    it's a directory */

```

```

                        AppendCharToHandle( dirH, ':' );
                    }
                    AppendCharToHandle( dirH, '\r' );
                }
            }
        }while( !done );

```

```

        DisposPtr( (Ptr)catPB );
        DisposPtr( (Ptr)wdPB );
        DisposPtr( (Ptr)fname );
    }
}

```



HyperChat™

A Look Into Networking



Donald Koscheka
Ernst & Young
MacTutor Contributing Editor
Volume 5, Number 12

Programming the Network

Network programming is fun. Over the past year, I've presented a suite of XCMDs that provide the Hypercard developer with network access. Rather than present a new XCMD this month, I thought it might be interesting to revisit some of the xcmts that we developed during the year. If you don't have access to back issues of MacTutor, you can obtain the source code for these xcmts from this magazine.

Programming the network is not unlike programming in Hypertalk. In both instances, you are concerned with the behavior of distributed processes. In HyperTalk, processes are distributed over several card and background objects. On the network, objects are distributed over time.

Time distribution of processes adds a new wrinkle to the programming task: how do you know when a given process has completed? The problem stems from the fact that we operate the network asynchronously; when we issue a request for some network service, we don't wait around for an answer. Rather, we return to Hypercard, leaving the network to handle the request in its own good time.

Figure 1 depicts a card for a prototypical network application called HyperServer. The card contains two icons, each of which provides some access to the network. Clicking on the volumes icon will cause the card to display a list of available servers. Selecting one of these servers will immediately send a message to the selected server requesting a list of its mounted volumes. Similarly, the catalog icon will return a catalog of the current folder on the server (for now, I leave it to your imagination to determine what other things one can do with such a card).

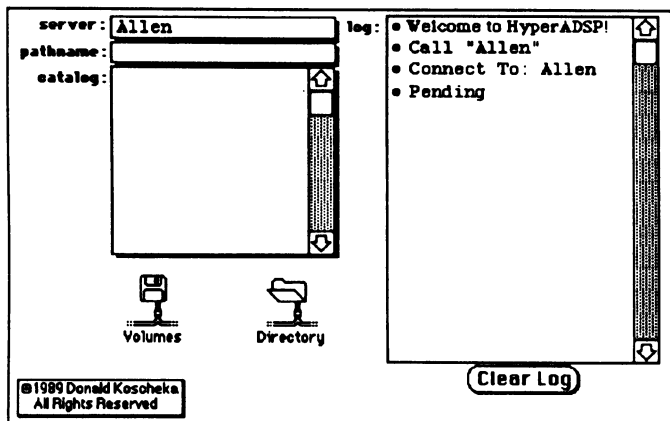


Figure 1. A prototypical screen for HyperServer

The process of sending a message to a server requires that you first call the server. Next you send a "get volumes" message to the server. Then you must wait for a response from

the server. Once the server responds, you hangup).

Waiting for some input before advancing to the next stage suggests a state machine. Figure 2 depicts just such a state machine. Each circle in the diagram is a state. The double circles are termini. They start or begin a process. The arrows are state transitions, their labels tell us what stimulus will force a transition to the next state. The looping arrows in the diagram suggest the concept of waiting. For example, when a call request is made, you can't immediately start talking, you have to wait for the other party to answer first. The process of answering the call triggers a transition to the connected state. Once connected, requests can be made to the server. Since requests imply a response, this state needs to wait for input before moving on. The process of receiving an answer to the request will trigger a transition to the next state (handle response). Once the request is serviced, we can go back to the connected state where we either wait for input and send another request (or hang up if that's appropriate).

State transitions can occur in response to one of two classes of stimuli: external or internal. External stimuli comprise information coming in over the "wire". Examples of external stimuli include service requests to a server and responses coming from a server. In figure 2, we model the external stimuli as the Service state and the Handle State. The handle state accepts the input from the remote end as a response to some previous request issued in the send state. For this reason, you should never get to the handle state unless you first pass through the send state.

The respond state in Figure 2 can only be reached from the service state. That is, you can only respond to a stimuli given that a request for service was made from a remote end. (For readability, not all states are depicted in the figure).

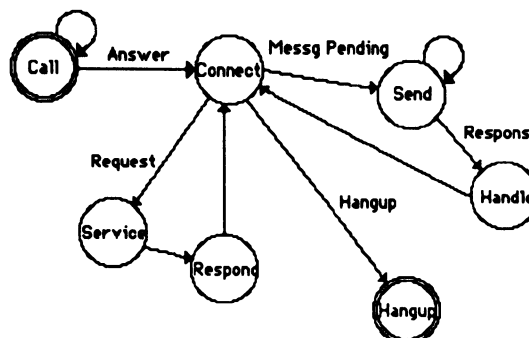


Figure 2 The State Machine For HyperServer

Internal stimuli are called requests. Requests are messages that are sent to a server. Once a request is issued, we must wait

for a response. For example, calling is an example of a message that request a connection with the server.

Once the server answers the call, we can move to the *connect* state. The connect state is implemented in the idle method of the stack method in listing 1. In addition to listening for external stimuli via ADSPListen, the idle method also manages the request mechanism.

We synchronize requests using a simple queue. Requests get added to the end of the queue and get serviced from the start of the queue. The queue is stored in the global container "message_queue". Each line in message_queue corresponds to a message. The first line in message_queue is the next message to service (not the current message). To execute a message, we first move it into the "message_pending" container. Every other element in the queue then moves up in the line.

The message_pending item will be executed on the next activation of the idle method. If a message is sent that requires some external response, it should be followed by a "PENDING" message. Pending does nothing which is why it's useful as a blocking mechanism. The only way for the queue to advance is to have some method activate the NextMessage method. So PENDING will remain the current message in the queue until some external stimulus unblocks it.

For example, to issue a call to a remote called "Allen" you might execute something to the effect of:

```
AddMessage "Call"&quote&"Allen"&quote
AddMessage "PENDING"
```

On the next pass through the idle method, the *Call* message will be executed invoking the call method in the script. The call method dials up the remote end and activates nextMessage which advance the queue to the next message, in this case the current message becomes "PENDING".

The queue cannot be unblocked until the server answers the call or the call times out. When the call is answered (in ADSPListen), the connectionMade method is activated unblocking the "PENDING" message.

Some messages have don't care results. When we hangup on a remote end, we generally don't care to wait around to see if the other guy hung up. Thus the HangUp message is not followed by a pending message.

As a rule, if you have a message that issues a request, you must follow the message with a "PENDING" message will tells the state machine to wait for an external stimulus before proceeding.

Listing 1 is the stack script that implements this state machine plus the response handlers (not shown in Fig. 1). Before running the state machine, we install HyperADSP in the openstack method. Similarly, we shut down HyperADSP in the closestack method.

Think of the idle method as implementing the *connect* state and the rest of the script should be a little easier to read. For brevity, I've left out the methods that originate the requests, but they're easy enough to reconstruct (or get a copy of this month's disk for a demo).

- Following are the stack methods for opening
- closing and listening for traffic on the
- network
-
- Due to space limitations, this stack is not
- complete. This month's source disk contains
- a complete demo of this stack.

```
on openstack
    global globalADSPData, globalSKTData, globalNBPData,
    myEntityName
    global HyperADSPData, adspcaller, LookupTable
    global message_queue, message_pending

    set the cursor to 4
    put empty into card field "current Server"
    put empty into card field "Subscribers"
    put empty into card field "Current Pathname"
    put empty into card field "Current Catalog"
    put empty into card field "log"

    put empty into message_pending
    put empty into message_queue

    if globaladspdata is empty then
        set the cursor to arrow
        adspinstall
        if the result is not empty then
            put the result
        else
            nbpOpen
            if the result is not empty then
                answer the result with "OK"
            else
                nbpRegisterName "", "HyperServer"
                log "Welcome to HyperADSP!"
            end if
        end if
    else
        log "HyperADSP already open!"
    end if
end openstack

on idle
    global message_pending

    -
    - if no message is pending, get one from the queue
    - if the queue is empty, do nothing
    -

    if message_pending is empty then NextMessage

    if message_pending is not empty then
        send message_pending
    end if

    adspListen "getTheMessage"
    pass idle
end idle

on getTheMessage
    global hyperadspdata, adspcaller, message_pending, incoming

    set cursor to 4
    set lockscreen to true

    put hyperadspdata into incoming

    get line 1 of incoming
    -
```

- Notice how we "turn the messages around" here.
- a put message tells us to take the incoming data
- a get message tells us to handle an incoming request

```
if word 1 of it is "put" then Receive incoming
if word 1 of it is "get" then Handle incoming
```

```
set lockscreen to false
set cursor to arrow
end gettheMessage
```

```
on closeStack
  adspRemove
  if the result is not empty then
    answer the result with "OK"
  else
    nbpClose
  end if
end closestack
```

- Network Callback methods.

-
- These methods are executed in response to
- some activity on the network
-

- connectionMade : answer an incoming call
 - doALookup: Update the names table
-

```
on connectionMade
  global adspcaller
```

```
  if adspcaller is not empty then
```

```
    - now that the connection is made, we can dismiss the
calling
    - message from the queue:
    Log "Connect to:"&&adspcaller
    put empty into adspcaller
```

```
    NextMessage
  end if
```

```
end connectionMade
```

```
on DoALookup
```

-
- - DoALookup returns a list of names along with
 - their registered types. There is a catch
 - that we need to handle: If we want to display
 - the name without the type, we need to delete
 - the last item in the line.
 -

- For example:
- Koscheka, Don, HyperServer
- needs to be reported as:
- Koscheka, Don.
-

- Note that asking for item1 of the line won't
 - do since the comma is part of the registered
 - name.
 -
-

```
global lookupTable, currentZone
```

```
set the cursor to 4
put GetZoneInfo( TRUE ) into currentZone
```

```
put nbplookupnames( "HyperServer", currentZone ) into
lookupTable
```

```
repeat with i = 1 to the number of lines in lookupTable
  get the number of items in line i of lookupTable
  put empty into item it of line i of lookupTable
  get the number of characters in line i of lookupTable
  if character it of line i of lookupTable is "," then
    put empty into character it of line i of lookupTable
  end if
```

```
  put line i of lookupTable into line i of card field
  "subscribers"
end repeat
set the cursor to arrow
End DoALookUp
```

- Generic utilities

- - SelectLine : return the content of the line
 - clicked in by the user (card field only)
 -
 - isBlank: return true if the field is empty
 - or contains just white space.
-

```
on log mess
```

- the idea for a network log is borrowed
- from Chris Allen.

```
repeat while the number of chars of cd fld "Log" > 15000
  delete line 1 of cd fld "Log"
end repeat
```

```
put "O"&&mess&return after cd fld "log"
end log
```

```
Function SelectLine theField
```

-
- select a line from the given card field
- Accepts the short name of a card field as input
-

```
put empty into my_answer
```

-
- When selecting a line from a field,
- you need to take into account that the
- field may have been scrolled.
-
- To determine the number of scrolled lines
- divide the number of scrolled pixels by the
- number of pixels per line
-

```
put the scroll of card field theField into box_top
put the textheight of card field theField into tsize
put round( box_top/tsize ) into lineNum
```

-
- You get a mouseup in global coordinates
- To find the line hit, you need to convert
- to local coordinates.
-

```
put item 2 of the rect of card field theField into
x_global_offset
put item 2 of the clickloc into x_hit
```

- Once you find the mousehit point,
- convert it to a line relative to top of box.
- This "relative line" is added to the number
- of scrolled lines to determine the actual
- line number (e.g. if 10 lines have been scrolled
- then the first line in the field is 11, not 1).
-

```
subtract x_global_offset from x_hit
```



```

add trunc( x_hit/tsize) to lineNum

-
- in the arithmetic, line numbers will be
- normalized to 0 rather than 1. Hypercard
- starts at line 1 so adjust the offset.
-
add 1 to lineNum
put line lineNum of card field theField into my_answer

return my_answer
end SelectLine

function is_blank string
  get character 1 of string

  if it is space or it is empty or it is tab or it is return
  then
    return true
  else
    return false
  end if
end is_blank

```

- Queue Management

```

-
- These routines handle management of the message
- queue. Excuse the logic; if Hypercard has an
- Achilles' heel, it's the lack of useful data
- structures.
-
- Addmessage : add a message to end of queue
- RemoveMessage: remove a message from the queue

```

```

on AddMessage the_memo
  global message_queue

  get the number of lines in message_queue
  put the_memo into line it+1 of message_queue
end addMessage

on NextMessage
  -
  - removing a message implies that the removed
  - message get put into the message_pending container
  - This method should only be called once the pending
  - message has been serviced
  -
  global message_queue, message_pending

  if message_pending is not empty then log message_pending

  put line 1 of message_queue into message_pending

  delete line 1 of message_queue
end NextMessage

```

- Message Management

```

-
- These routines handle management of the possible
- messages that can be received by the network.
- all messages have the form:
-
- Line 1: Message Type (token)
- Line 2: Message Sender (string)
- Line 3..n: Message dependent data
-

```

```

Function BuildHeader mess, name

```

```

  put mess into line 1 of temp
  put name into line 2 of temp
  return temp
end BuildHeader

on call someone
  adspCall someone
  NextMessage
  - notice that this message needs to be pended also
  - The block will be cleared by the connectionMade method
end call

on hangup someone
  ADSPHangup someone
  NextMessage
end hangup

on PENDING
  - This is a "wait for completion" message. Because
  - it does nothing, it blocks the message_queue until
  - some data is received. The receive method will clear
  - the pending message. Only the nextMessage method can
  - unblock the queue. Notice that nextMessage should get
  - called only after the expected data is returned.
end PENDING

on getcatalog server, folder
  global myEntityName
  -
  - Client is asking for a directory of the requested
  - folder. Service the request...
  -
  if server is not empty then
    put BuildHeader( "get catalog", myentityname) into
    send_this_message

    put folder into line 3 of send_this_message

    ADSPTalk server, send_this_message

  end if
  NextMessage
end getcatalog

on getVolumeList server
  global myEntityName

  if server is not empty then
    put BuildHeader("get volumes", myEntityName ) into
    send_this_message

    Log server
    log send_this_message

    adsptalk server, send_this_message

  end if
  NextMessage
end getvolumeList

-
- Message request methods. Requests are assembled
- by these methods and added to the request queue
-

```

```

on request_volumes
  global message_queue

  put selectline( the short name of the target ) into the-
  Server
  if theServer is not empty then
    put theServer into card field "current server"

```

```

AddMessage "Call"&&quote&theserver&quote
AddMessage "Pending"
AddMessage "getVolumelist"&&quote&theserver&quote
AddMessage "Pending"
AddMessage "Hangup"&&quote&theserver&quote
end if

end request_volumes

on request_catalog
  global message_queue

  put card field "current server" into theServer
  if theServer is not empty then

    AddMessage "Call"&&quote&theserver&quote
    AddMessage "Pending"
    put quote&cd fld "current pathname"&quote into temp
    AddMessage "getcatalog"&&quote&theserver&quote&","&temp
    AddMessage "Pending"
    AddMessage "Hangup"&&quote&theserver&quote
  end if

end request_catalog

```

-
- Message response methods. These methods will
 - respond to a particular message from a remote
 - end in whatever manner is appropriate. Note
 - that responses don't get queued, they go
 - right out on the line.
 -
 - Response messages take the same form as request
 - messages. Notice that the response methods
 - use myentityname to tell the client which server
 - is providing the response.
-

```

on Receive something
  -
  - A request that was made in the past is
  - being addressed, do something with the
  - incoming data (for now, just display it)"
  -
  if "volume" is in line 1 of something then
    log something
  end if

  if "catalog" is in line 1 of something then
    delete line 1 of something
    put line 1 of something into cd fld "current server"
    delete line 1 of something
    put something into cd fld "current catalog"
  end if

```

```

NextMessage
end receive something
on Handle request
  -
  - incoming caller is making a request, do
  - something about it
  -

  get word 2 of line 1 of request
  put line 2 of request into to_client

  if "catalog" is in it then
    put line 3 of request into of_folder
    return_catalog of_folder, to_client
  end if

  if "volumes" is in it then
    return_volume_list to_client
  end if

end handle

on return_catalog folder, client
  global myEntityName

  if client is not empty then
    put BuildHeader( "put catalog", myEntityName ) into
    the_response

    Put GetCatalog( "", directoryID ) into temp
    put the_response&return&temp into the_response

    ADSPTalk client, the_response

  end if
end return_catalog

on return_volume_list client
  global myentityname

  if client is not empty then
    put BuildHeader( "put volumes", myentityname )>
    into the_response

    Put Volumes() into temp
    put the_response&return&temp into the_response

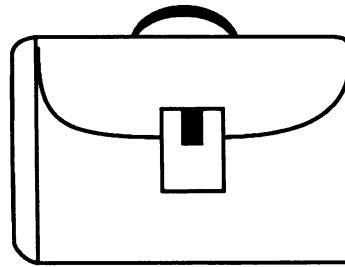
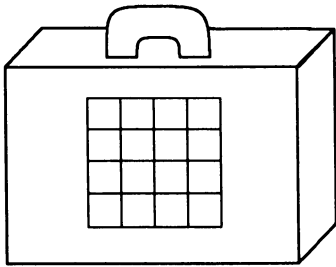
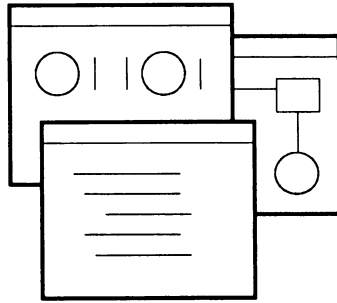
    ADSPTalk client, the_response

  end if
end return_volume_list

```

Listing 1. Stack Methods for HyperServer

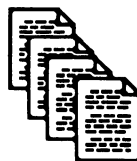




More Articles

Resource Roundup

20 Steps to Printing Incompatibility



Joel West
Palomar Software
Oceanside, CA
Volume 5, Number 6

20 Steps to Printing Incompatibility

Incompatible printing code. Based on what I've seen, this is a popular pastime among Macintosh software developers.

First, some background. I'd like to apologize to *MacTutor* readers for the long delay between articles. Things were pretty busy in 1988 as we worked on our printer drivers, culminating in the January announcement at Macworld Expo of drivers we developed for the HP PaintJet and the Howtek PixelMaster.

For an application writer, the two drivers represent two more examples of output devices to be compatible with. The PaintJet 1.0 supports 2, 8, and 256 (fixed) colors with bitmap fonts, while the PixelMaster 1.0 is an 8-color and 24-bit color driver with the Mirus/URW outline fonts.

There are many other Chooser-level printer drivers being offered out there. If you count just those drivers that are sold with a printer (see Table 1), the number has exploded from two years ago, when there were two Apple printers and no third-party solutions. There are many more (I've never been able to keep up) independent drivers, from companies like Phoenix (née Softstyle) and GDT.

Of course, it's not possible for every application to test with every printer driver for incompatibilities, which brings me to one of the main points of this article.

Device Independence

Many programmers think you have to know what printer you're sending to. Each printer has its own characteristics, the argument goes, and it's important to optimize for those characteristics.

Nine times out of ten, that's plain wrong. The Macintosh printing architecture is designed to be device-independent, and, though it has its weaknesses, in that regard it is pretty successful.

I'm certainly not part of a Thought Police that forbids you from such practices. It's just that one of the largest areas of application/printing incompatibility comes from device-dependencies.

It's particularly frustrating because these problems are almost always unnecessary. Perfectly good driver-independent solutions are available for rotated text, smoothed polygons, custom line layout, determining paper size, and other problems described later on.

And there is no question that the problems introduced by such device-dependencies, on the balance, outweigh the benefits of same.

Read the Documentation

The second easy way to avoid problems is to restudy the documentation, beginning with the "Printing Manager" chapters

of *Inside Macintosh*. Too many people don't even understand these basics, as evidenced by some of the problems below.

A sample printing loop is shown in Example 1. Using such a loop (or something very similar) will, unfortunately, rule out at least five of the incompatibility opportunities below.

When doing your basic research, don't forget that the rules have changed over the years.

There's PrGeneral functionality in *Inside Macintosh*, Volume V. Among the tech notes, I'd recommend #92, 118, 125, 149, 161 and 192 for compatibility, and #72, 73, 91 and 95 for extra functionality.

There are many applications that want to get the printing code to do something that is not in the published interfaces. I understand this problem well and am certainly sympathetic; perhaps three years ago I got into this debate on Usenet, arguing the plight of the application developer. One of the examples I used was for a painting program to set the Imagewriter (or other drivers) to "square pixels" or the equivalent.

Unfortunately, if you're not following the documented interfaces, how can you pull it off? Usually the answer is trial and error iteration. It may work with the two drivers you try, but there will be 10 or 20 more that do don't try, and many of these will break.

Today, one of the better solutions is to provide preformatted stationery with configured print records for specific printer drivers. Another approach would be to allow the user to set a default print record once, and then use that print record from then on out.

If you're not sure your approach is compatible, I'm sure ZZ or Luke at tech support would be glad to offer advice on a specific trick or approach. As always, link to MacDTS.

20 Easy Steps

However, you may decide that you really want printer incompatibilities. Perhaps you're trying to sell a particular brand of printer. Or maybe you need a crisis to convince management to hire a large tech support staff.

Or maybe if your sales are poor enough, your publisher will go bankrupt and you can reclaim your copyright to market a better version of your application.

Based on Palomar's research, and with assistance from other driver writers, we've found that such incompatibility is very easy for you to achieve.

To make things more complete, I've decided to mention specific examples where I had them. This is not to reward these particular programmers or companies, but just note the ones that happen to be on my list.

To my knowledge, all of these problems are known to the

application developer and most are either fixed in later releases, or in the process of being fixed.

The list is heavily skewed to the major applications, because those are the ones Palomar tests with most extensively. This doesn't mean that these applications are more successful in obtaining incompatibility; nay, we expect that there are many more unheralded applications on the second 50 of a top 100 list that are equally worthy.

Finally, in defense of my fellow programmers, it should be noted that the Macintosh printer architecture has evolved in a somewhat *ad hoc* fashion since the Imagewriter was introduced in 1984. As such, most of the decisions listed below reflect reasonable guesses based on an incomplete specification; many would qualify as outdated assumptions more than deliberate perversions.

With that, let's get on to the list of easy incompatibilities.

1. Assume only two printers

Example: FullPaint

To my knowledge, only one application checks the `wDev` field and puts up the alert "could not find the print driver." This unique code looks something like this:

```
if (wDev >> 8 == bDevLaser)
{
    /* print one way */
}
else if (wDev >> 8 == bDevCItch)
{
    /* print another */
}
else
{
    Alert(ALRT_YouLoseSucka, NULL);
}
```

However, many (usually older) applications have code that looks like:

```
if (wDev >> 8 == bDevLaser)
{
    /* do nice printing */
}
else
{
    /* do cruddy 144 dpi printing */
}
```

There have been more than 20 different printer drivers allocated a distinct `wDev` value. Just because it isn't a LaserWriter, doesn't mean that it will produce cruddy output.

2. Special case on printers

Example: Digital Darkroom 1.0, XPress 2.0

This is slightly more subtle than the previous approach, and offers many opportunities for incompatibilities:

1. If new printers are introduced, you get lousy results with them.
2. If existing drivers are improved, you get the same results as with the old driver.
3. If the implementation of the driver changes, you will likely break if you use device-specific information (such as the layout of the print record).

There is one common special casing that has produced less

incompatibilities than most:

```
if (wDev >> 8 == bDevLaser)
{
    /* send PostScript stuff */
}
else
{
    /* send QuickDraw stuff */
}
```

However, this will cause problems with a save-now, print-later utility like Glue or OpenIt.

Many, many applications go well beyond this PostScript/non-PostScript distinction to achieve printing incompatibility. They use all sorts of intimate details about the driver, what comments its supports, etc., etc. In several leading applications, the author has gone to the trouble of building tables with device-specific information so all sorts of things will be handled differently.

As will be examined below, in most cases there are other ways to accomplish the same thing. For example, you can special case the paper size based on the `wDev` (incompatible) or read the `prInfo.rPage` and `rPaper` fields of the print record (compatible).

For many applications, this is the preferred approach to printing incompatibility. I've picked Digital Darkroom because the incompatible and compatible approaches are about the same difficulty.

In particular, DD has a better halftoning mechanism that it only makes available to the LaserWriter, LaserWriter SC, and GCC PLP. This means that any other high-resolution black & white printer is out of luck.

One better approach would be to always make this advanced halftone available. A second, more restrictive approach, would be to pick a threshold resolution, (say 220 or 300 dpi), and if a `PrGeneral()/getRsl` call returns the maximum or device resolution as higher than that, then make the option available.

3. Bypass printer dialogs

Examples: HyperCard 1.2, MPW 3.0

This is the last of the items that every printer driver writer mentions.

It is the bane of our existence, because every time we add functionality to a driver, the application takes it away (or makes it difficult to find). Many printers have a print quality selection, or paper feed option; some have a spool for later printing options; our drivers include a preview before printing options. Try addressing a fax without the fax dialog some time.

HyperCard and MPW are two applications that do not show the job dialog for the "Print" command.

I should also note that Word 3.0 makes it a challenge to get at the dialogs (something that is fixed in 4.0.) PageMaker 3.0 has a non-standard interface that only gives you the style ("Page Setup") dialog after you select printing.

From a personal standpoint, I find the hierarchical menus in Cricket Presents 2.0 to be somewhat frustrating and confusing. On the other hand, I've long bitched that MORE 1.0 should print bullet charts in landscape and outline views of the same document in portrait, so I guess Cricket has a creative (if unusual) way

to provide needed functionality.

4. Spool-a-page, print-a-page

Example: MacWrite 5.0, XPress 2.0

Once upon a time, people tried to print large files from floppy-based systems. Thus, some applications would submit print jobs to the printer one page at a time.

This is no longer necessary, but it does achieve incompatibility by defeating the functionality of many printer drivers. As noted in *Macintosh Technical Note #125* this is a good way, for example, to confuse print spoolers.

The counter approach is for the driver to lie and always claim it is "draft" printing. Unfortunately, this reduces the performance of more normal applications, which may not be expecting to free much memory for draft printing.

5. Directly change GrafPort fields

Example: SuperPaint 1.0, FullWrite 1.0

There are several reasons why this is a great way to achieve incompatibility just on general principles.

But we found that when you supply a CGrafPort for color printing to an application, there are a few "bit bashers" that provide frequent introduction to "la bomba."

For example, if you directly change `grafptr>bkPat` (as does SuperPaint 1.0) then, on a CGrafPort, you are clobbering the `cgrafptr>bkPixPat` handle. If the address is weird enough, like `0xFFFFFFFF` (can you say "black"), you will get a bus error.

Those GrafPort fields that are used for other fields in a CGrafPort include:

- `portBits`
- `bkPat`
- `fillPat`
- `pnPat`

Change any of these directly and you're almost certain to achieve that desired level of incompatibility.

The bottom line to achieving incompatibility is to make sure your field manipulation will be invalid on either a GrafPort, Color QuickDraw CGrafPort, or a 32-Bit QuickDraw CGrafPort. The 6.0 LaserWriter driver is one way to test this, as are various drivers from Tek, Palomar and the film recorder companies.

6. Send bitmaps instead of text

Example: CricketDraw 1.1, FreeHand 1.0, HyperCard 1.2

This one actually is more work than working compatibly. After all, the Font Manager knows how to draw text, if you just use `DrawText()` and `DrawString()` instead of `CopyBits()`.

Also, no one is going to do this on the LaserWriter – every customer will notice it immediately – so you must special case out this approach on the LaserWriter. Thus, it's even more work to do `CopyBits()` for just the other printers.

My favorite example is the Cricket Expressions drivers with built-in URW outline fonts. Because CricketDraw sends only

bitmaps, those fonts are not available and you get those lovely low-res bitmaps.

This one is a real opportunity to put your worst foot forward. Can you imagine how lousy 72 dpi text looks on a 4,000-line film recorder or 300 dpi printer? Can you imagine any serious business professional using your application on such an output device? Your place in incompatibility heaven is assured.

7. Send bitmaps instead of rotated text

Example: MacDraw II 1.0

In this case, it's easier to be incompatible than compatible, but it's not strictly necessary.

The original designers of MacDraw (1.9) came up with a clever way of hiding both the bitmap and text representations of rotated text. This means that a smart printer driver (e.g., the LaserWriter) can get at the actual text and its angle, while a dumb printer driver (e.g., ImageWriter 2.7) will automatically print the bitmap representation.

The compatible way is to use the `picTextBegin` and `picTextEnd` comments, as documented by *Macintosh Technical Note #91*.

How does this work? You send both the bitmap and text representations, using an empty clipping rectangle to "hide" the text from any dumb printer drivers, as below:

```
PicComment(picTextBegin, begh);
PicComment(picTextCenter, ctrh);
GetClip(saveclip);
ClipRect(gZeroRect);
DrawString(theText);
SetClip(saveclip);
CopyBits(theBits);
PicComment(picTextBegin, NULL)
```

MacDraw 1.9.5 did it this way, but the MacDraw II assumed that any printer that's not a LaserWriter is brain-dead. However, the good folks at Claris don't seem to appreciate incompatibilities, so it's back to the compatible approach with MacDraw II 1.1.

8. Smooth polygons instead of comment

Example: FullWrite 1.0, MacDraw II 1.0

Again, the incompatible way is much easier, but there is a clever MacDraw/LaserWriter convention for compatibility with all drivers.

The approach is very similar to the rotated text case: you can provide both types of information and let the driver decide.

One piece of information is the smoothed polygon decomposed into individual line segments at a specific resolution. The other is the set of original inflection points, so that the polygon can be re-rendered at any resolution.

9. Send bitmaps instead of objects

Example: Excel 1.5

The final example along these lines is sending screen-resolution bitmap shapes instead of their QuickDraw equivalents.

This is a great way, for example, to avoid drawing a circle

marker on a line plot at the actual resolution of the printer. You can output a 72 dpi bitmap instead of a 300 dpi shape, thus helping the user to conclude that \$6,000 on a laser printer was money down the drain.

10. Secretly do own line layout

Example: ReadySetGo 4.5, PageMaker 3.0

Some applications want to do their own line layout. There are certainly good reasons to do that, although I would hope someday to have a standard system routine to handle this the same way for every application and printer driver.

The incompatibility opportunity is to do your own line layout *secretly*. Don't send a `picLineLayoutOff` comment (#155), as described by *Macintosh Technical Note #91*. Then your application can fight the printer driver for control of the text placement.

11. Assume paper size

Example: PowerPoint 2.0, MacWrite 4.1

There are lots of assumptions that can be made about "paper" size, many of them offering incompatibilities.

PowerPoint assumes that the top and bottom margins are the same, so it centers things within the imageable area. If the printer has a larger top or left margin, then the output will not be centered on the physical page.

The original MacWrite assumed that a piece of paper could not be wider than, say, 8" or so. This achieves incompatibility many oddball choices, such as printing in landscape mode.

12. Use large drawing coordinates

Example: Studio Session

QuickDraw helps with this incompatibility, since it limits drawing coordinates to 16 bits, or $\pm 32,767$. But it does take a little unusual cleverness to find this one.

Suppose you're drawing a horizontal line on a page of a 300 dpi QuickDraw printer. You could draw from (72,0) to (72,576), say, to draw an 8" line. When scaled up to 300 dpi resolution, this maps to (300,0) to (300,2400).

Of course, you could also draw the same line as (72,0) to (72,28800). After all, everything outside the page will get clipped, right?

At 300 dpi, this gets scaled to (300,0) to (300,120000). Since 16 bits won't hold the latter coordinate, you get (300,11072), and your line magically disappears! Neat, huh?

13. Hog or fragment memory

Example: Excel 1.5

Excel has its own unusual memory management; others may also be unusual, but none so among the Top 10 best-sellers.

The current Excel asks for a large amount of nonrelocatable memory, not leaving much memory available for the printer driver. This is the single reason many printer drivers are aggressive about demanding memory from an application.

Taking lots of memory, or allocating large nonrelocatable blocks are reasonable ways to achieve printing incompatibility.

At the same time, the driver may create fragmentation

problems for your application that also show up for incompatibilities.

If you want to be incompatible, then assume that `SetPtrSize()` will always work on a nonrelocatable block. It will often fail due to heap fragmentation, such as a call to `MoreMasters()` during printing or even an open desk accessory.

If `SetPtrSize()` does fail, you could instead allocate a new nonrelocatable block and then `BlockMove()` the data over.

14. Barf when memory stolen

Example: Full Impact 1.0

Nowadays, drivers that use QuickDraw imaging require lots of memory. At 300 dpi, a black & white 8" by 10" page requires 879K, while 240 dpi x 8" x 10" by 24 bits of color requires 13 Mb. And, of course, outline fonts require lots of code, temporary memory and decent caches for adequate performance.

With some the memory management scheme of some applications, such a driver/application combinations produces wonderful incompatibilities.

They may very well push your grow zone procedure and request more memory than currently available in the heap, particularly in light of item 13 above.

For a draft-mode printer, this may come at the time of `PrClosePage()` or `PrCloseDoc()`. For a spool-mode printer, this normally comes at the time of `PrPicFile()`.

Delightfully, some applications immediately panic if their grow zone procedure is called. Many are following the advice from page 64 of the first edition of *How to Write Macintosh Software* says that

In the worst case, when your grow zone gets a memory request that you can't fulfill, even by eliminating the monitor object, you should force the user to save any work in memory and then quit the application.

Since Scott Knaster never articulated a goal of printing incompatibility, it may be assumed that this is an outdated approach to compatibility rather than part of a deliberate incompatibility strategy.

If you're not after incompatibility, then a reasonable algorithm to use is the following:

1. Stash some memory for later use by the grow zone.
2. If the grow zone is asked for memory during printing, give it up; if you can't satisfy the request, just say "no" (i.e., return a size of 0).
3. At the end of the printing loop, try to reallocate the memory reserve.
4. If you then have no reserve, complain and quit.

15. PrOpen() at beginning of application

Example: MacDraw 1.9.5

This historically correct approach is now a great way to achieve incompatibility. In particular, it causes nifty problems with the Chooser and MultiFinder.

If you're trying to be boring and compatible, then do your `PrOpen()/PrClose()` around each group of printing calls, as shown in the example and recommended by *Macintosh Technical Note #161*.

16. Change print record directly

Example: PixelPaint 1.1

The print record structure is fairly well-documented and well-hacked. (I should know: I contributed to the documentation with my March 1987 "Printer Sleuthing" article.)

However, every driver treats these fields slightly differently, offering many opportunities for incompatibilities. A trick that will fool one driver may break another.

The one I found most effective was in the original Pixel Paint, which (as with some other incompatibilities) forces print record values to fix one bug in one version of a printer driver. The end result was to prevent landscape printing for most drivers.

Many applications obtain incompatibility by forcing paper sizes directly by changing the print record. This is usually quite effective, because you usually don't know what you're doing, and the driver won't either.

17. Send fake handles

By "fake" handle, I mean a pointer to another pointer that is not a master pointer.

I don't have the specific application example in front of me, but this is just as useful with the Printing Manager and printer drivers as it is with the entire Toolbox. Such a synthetic handle will effectively confuse the Memory Manager if the printer driver should be so foolish as to pass it to the Memory Manager.

Macintosh Technical Note #117 gives you lots of great ideas on how to do this and why this works.

18. Don't call PrPicFile

For this one I don't have the application either; however, it's very easy to explain.

The standard print loop includes a check for spool-mode printing:

```
PrCloseDoc(myprport);
if ((*ph).PrJob.bJDocLoop ==
    bSpoolLoop && PrError() == noErr)
{
    GetRidOfLotsMoreMemory();
    PrPicFile(ph, NULL, NULL, NULL, status);
};
```

For some reason, some applications achieve incompatibility by not calling PrPicFile() all the time. This is, after all, a needless step for the LaserWriter and other draft-mode printers.

If you don't call PrPicFile(), then no output will be produced; the effect can be quite mystifying and is usually good for one phone call per customer.

19. Ignore printing errors

Example: Reflections, Cricket Presents 2.0

If you want your program to halt and crash unpredictably, you can just blindly ignore Memory Manager and Resource Manager errors. Similarly, printing incompatibility can be achieved by ignoring printing errors.

There are many different places that an error can occur during printing, such that a compatible program would call

PrError() after every Printing Manager call.

The most common errors an application will see are:

128	iPrAbort	user cancelled printing
-27	iIOAbort	cancelled after printer not responding
-108	iMemFullErr	not enough memory to print

There are several other errors you can get. For example, many File Manager errors are possible while spooling to disk.

Cricket Presents is very creative in using this to achieve incompatibility: it crashes if the disk fills up when spooling to disk. Cancelling or other errors returned by PrPicFile() are just fine.

One error you probably won't see is iIOAbort, since the printer driver resets iIOAbort to noErr after PrPicFile(). This means that if you try to decode error messages before the end of the printing loop, you'll have the opportunity to display a few spurious error messages.

20. Ignore page number selection

Example: Persuasion 1.0

The original *Inside Macintosh* explains the standard suboptimization for imaging only those pages that are necessary. As explained on page II-156, look at the page numbers in the print record, pass only those to the printer driver, and convince the driver to print all the pages you pass it.

Almost all of the commercial applications do this. For example, this approach is built into MacApp.

What are the incompatibility opportunities here? Your application may end up imaging 20 or 30 complex pages in order to print just one. Not only can you tax the patience of the poor saps who bought your software, but you can also fill up their hard disk, preventing them from running it at all if they don't own a brand-new 80 Mb drive.

Credits and Disclaimer

I'll take the blame for divulging secrets but I can't take all the credit for finding problems.

The rest of the development team at Palomar deserves credit for much of what we found: Neil Rhodes, Mark Anderson, Michael Phil Ogawa and Marshall Clow. Our testing expert, Julie McKeehan, found many of the problems for the first time.

The folks at HP did a lot of pounding, too. Although full names are not HP policy, credit goes to Bill, Terri, Cindi, Bob, Robin, Dave and Marty, among others.

There are quite a few other driver writers out there who helped me, but they seemed less interested than I in awarding examples of incompatibility, probably because some dunderhead managers at the various application houses won't realize that this whole article was a tongue-in-cheek look at a serious problem. So to Scott, Mark, Jacques and Seth, thank you for your support – and refer any irate calls to me when this hits the stands.

Example 1: Standard printing loop

```
PrOpen();
err = DoPrinting(docprinthand);
```



```

PrClose();
...
OSErr DoPrinting(THPrint ph)
{
    Integer firstpage, lastpage, numpages;
    Boolean isspool;
    GrafPtr saveport;

    if (!PrJobDialog(printh)) return kDidNotPrint;

    GetPort(&saveport);
    /* Suboptimize page nos. */
    firstpage = (**ph).PrJob.iFstPage;
    lastpage = (**ph).PrJob.iLstPage;
    numpages = lastpage - firstpage + 1;
    /* fool printer driver */
    (**ph).PrJob.iFstPage = 1;
    (**ph).PrJob.iLstPage = numpages;

    isspool = (**ph).PrJob.bJDocLoop == bSpoolLoop;
    if (isspool) numpasses = 1
    else numpasses = (**ph).PrJob.iCopies;

    FreeUpSomeMemory();
    myprport = PrOpenDoc(ph, NULL, NULL);
    for (copyno=1; copyno<numpasses; copyno++)
        for (pageno=firstpage;
            pageno<lastpage && PrError()==noErr;
            pageno++);
        ( PrOpenPage(myprport, NIL);
          If (PrError() == noErr)
              DrawAPage(pageno);
          PrClosePage(myprport);
        ); /*each page*/
    PrCloseDoc(myprport);

    /* Now print spooled document */
    if (isspool && PrError()== noErr)
    {
        GetRidOfLotsMoreMemory();
        PrPicFile(ph, NULL, NULL, NULL, status);
    }
}

```

```

});

SetPort(saveport);
TakeBackSomeMemory();
/* Now see if there were any errors */
errno = PrError();
if ((errno <> noErr) &&
    (errno <> iPrAbort)) /* ignore command-period */
{
    CallErrorHandler(errno);
    return errno;
}
else
    return noErr;
}

```

Manufacturer	Printer
Apple	AppleFax
Apple	ImageWriter LQ
Apple	ImageWriter
Apple	LaserWriter II SC
Apple	LaserWriter†
GCC	PLP
GCC	WriteMove
Hewlett-Packard	PaintJet
Howtek	PixelMaster
Mirus	FilmPrinter
Presentation Tech.	Montage FR1
Tektronix	4693D
Tektronix	ColorQuick

† Many other PostScript® devices are also supported by this driver.



Editorial

Developer Conference Report



David E. Smith
Editor & Publisher
MacTutor
Volume 5, Number 7

System 7.0 Detailed

Apple Computer Inc. previewed its new System 7.0 software direction to an estimated 1,500 software and hardware developers at the Worldwide Developers' Conference in San Jose on May 9, 1989. The new version represents what Apple calls a "feature release", which means Apple wants all users to upgrade to this version. It will be compatible with all Macintosh computers from the Mac Plus and up, but only 68030 based machines or those using the 68020 with a PMMU will have virtual memory capability under System 7.0. This version will shut the door on the older Mac 128K and Mac 512K machines, making the Mac Plus the official bottom of the product line, if it isn't already. To stay in the game, all users should plan on upgrading their Macs to the functionality of a Mac Plus. Beyond that, there is the question of whether a shop should upgrade all their machines to at least a 68030 based machine. Both the SE and the Mac II offer upgrade paths to the corresponding SE/30 and Mac IIfx machines.

Go Buy Some RAM

In addition to having at least a Mac Plus machine, System 7.0 requires at least two megs of RAM to run. So plan now on attending a MacWorld Expo and buying more memory. Show prices typically run \$150 per megabyte. Future Macintosh computers may have a ROM upgrade as well, allowing them to work with System 7.0 with only one meg of memory by putting more of System 7.0 in ROM. Both the SE/30, IIfx and IIfx have ROM SIMM card slots to allow for a future ROM upgrade. Apple intends to move its entire product line to System 7.0 so they will find a way to make it work as cheaply as possible for their production units.

New Mac Rumors

The Apple press release for the conference makes mention of the fact that "Apple expects to continue to offer an attractive low-end product with System 7.0." We leave it to the reader to guess what that means. Several comments were made that a low-end Macintosh is coming but it is hard to imagine what that might be although rumors suggest Apple has already figured out how to improve on the Mac IIfx and make it cheaper to build. Rumors have also suggested a new high-end Mac IIfx running at 25 MHz is due shortly. And the fact that the next Mac World Expo in San Francisco has been pushed back from January to next April suggests that Apple has some great new product to release that will not be ready until April. Put all this together and you can probably conclude the following:

- New low-end Mac at the Boston Expo?

- New high-end 25 MHz Mac IIfx Fourth Qtr.?
- Lap-top Mac at the San Francisco Expo '90?

In the next twelve months Apple will release a low-end Macintosh and a 25 Mhz Mac IIfx of some sort. These two products will compliment what Apple is calling "the mid-range" of the product line, the heart of their expected sales in the next 18 months, the Mac IIfx and the SE/30. Then in April, figure that Apple will release the long-awaited lap-top Mac. But if the portable comes out as a wimp machine with only a 68000 in it, then it will bomb. It better be at least a 68020 with a PMMU or it won't fly. The world does not need a battery Mac. The world needs a moveable Mac that really does fit under an airline seat!

Core Technologies

The main parts of System 7.0 (read the stuff that Apple is committed to getting done on time) are the virtual memory, IAC, outline fonts and improved printing. The memory manager and other system software have been re-written to be 32-bit clean so that System 7.0 will include full 32-bit addressing. This will allow users to plug in 4 megabyte SIMM boards into the Mac motherboard for 32 megs of on-board RAM. Amazing isn't it? The physical memory limit will move out from the current 8 megs to 128 megabytes of RAM with a virtual address space of up to 4 Gigabytes. To make sure your applications are "32-bit clean", see technical note #212 & 213. Basically, you don't go around the memory manager by manipulating handles directly or trying to use the top 8 bits in a handle you have assumed to be only 24 bits of address.

Some of the "fixes" Apple had to do to the memory manager to make it 32-bit clean was to move the flag bits out of the top end of the master pointers, change the block header format and size, and change the zone header format and size. They also had to change the interface to the CalcCRegion in Quickdraw for custom controls (CDEFs), as well as clean up various OS, toolbox, Finder and MultiFinder problems. A lot of late-night soda pop didn't hurt either!

One interesting note about System 7.0 is that Apple will no longer special case the operating system for Microsoft software that can only run in one megabyte of address space. So older versions of Microsoft products may stop working under System 7.0 because the Mac will no longer check and load those products into the first meg of memory. The fact that Apple did this checking at all is a testimony of the importance of the Apple / Microsoft relationship when the Macintosh was first introduced. Obviously, Apple feels less dependent on Microsoft and is no longer willing to put up with their software constraints.

Virtual Memory

Virtual memory will extend the system RAM to typically double whatever RAM memory the user has, allowing more practical use of smaller memory machines. This will be invisible to the user; he won't have to worry about configuration or anything, it will just happen. As mentioned before, this feature will only work on 68030 machines or Mac II's with a PMMU. The applications will "see" this memory as an extension of normal RAM space so no changes will be required to the applications to take advantage of virtual memory. Going out to the disk will decrease performance slightly and Apple is suggesting that a reasonable setting for the virtual memory is twice the RAM space to prevent a lot of disk thrashing.

Under System 7.0, there will no longer be a Finder versus MultiFinder choice. MultiFinder will be "on" all the time. The VM model is a true demand paged (4KB) model. In a future system release, perhaps 8.0, it will also feature protection and a typical mini-computer file system and preemptive scheduler (i.e., true multi-tasking). Incidentally, the new Finder was written in C++ (it works on my machine!). Apple has about 200 authorized copies they are using internally, but they have not received the final 2.0 version from AT&T to yet make it available to developers. As a result, C++ is not expected to be available to developers outside of Apple until September. It will support MacApp however, when it is released.

IAC

Apple is finally including IAC in System 7.0. The implementation is not too unlike the one published in MacTutor last year by Frank Alviani and Tom Snively and Frank will have an expanded article on it in an up-coming issue. The main thing that the IAC gives you is a standard mechanism for applications to pass messages back and forth. It will also support dynamic linking of documents with a live "copy/paste" mechanism. Apple will define a set of standard messages called "AppleEvents" that applications can use to request actions of another program, such as "open document".

To implement AppleEvents, the event manager has been changed to allow a high-level interface for program to program communications by applications. The event record data structure has been expanded. The mouse *point* has been changed to be a LongInt. If the *what* field indicates a *HighLevelEvent*, then *message* field and *point* field now indicate a class and type of message. New toolbox calls allow applications to post and accept these new "high level events" as they are called. An application can send messages to the Finder, for example, to print a report. Non-event type code (I/O drivers) can also communicate with applications with a low level interface to PPC services. If applications want to implement their own message protocol beyond AppleEvents, then they must register their protocol with DTS just as the creator tags are registered, so as to avoid mass confusion.

What's That You Say?

The key to getting application-to-application communica-

tions to work smoothly is that everyone must speak the same language and that requires that Apple set the standard by defining what we all should be able to respond to. We are still waiting for Apple to define what is meant by "formatted text" but hopefully they will do a better job with IAC. The magic of the Macintosh was not the fact that you can cut and paste between documents, but rather, that Apple defined a standard TEXT and PICT data type that allowed multiple vendors to design tool applications that could read and write compatible documents. That standard data type needs to be extended to include formatted text and other data types as well as standard commands, or AppleEvents as they are being called. We encourage Apple to continue to lead the way in defining standard data types for both text and graphics as this will make the design of meaningful IAC applications much easier.

One of the important messages of the conference concerns something not officially talked about yet. Apple plans to define an official "Apple Script" based on HyperTalk and put it into the system. This will allow a common "scripting language" that any application can use, including the Finder. It means that developers can add user extensibility to their applications using Apple Script without having to write a parser or compiler into their program. At least that is how I hope it will work!

The Postscript Wars

Apple has declared war on Postscript with the announcement that Quickdraw will continue to be the imaging engine on the Macintosh, and in fact, will continue to be enhanced so that it is better than Postscript. The first salvo from that commitment is the release of Quickdraw outline fonts, a Layout Manager for kerning, and a new printing architecture. This means that there will be only one type of font in the system file from now on: outline fonts. But they will come in two varieties: Apple outline fonts or Postscript outline fonts. The new printing manager will download both types of outline fonts to the LaserWriter so that the user will not need to know which type of fonts he has in his system file. Both will get printed at any resolution in the same quality.

Outline fonts are defined by a quadratic curve and a stack based language instruction set. It is designed to be fast and flexible to allow for international extensions, and easy translation from other font formats. And again, this cryptic remark: "flexible, not limited to type; designed for future System Software". What does that mean? Probably tied into future Quickdraw enhancements somehow.

Other technical features include projection and freedom vectors, automatic angle adjustment and a delta instruction for pixel placement at the smallest sizes. In Postscript parlance, I think this is called "hints". The size of the "dot" can also be adjusted based on the print engine being used, which is important for high end typesetting. Other typesetting features include a Line Layout Manager that draws text by using a "layout" that describes kerning, justification, fractional position, optical alignment and other features important for easy support of non-Roman languages. Apple positions the Line Layout Manager in

quality and complexity somewhere between TeachText and Page Layout, whatever that means. In addition to drawing text by layout, it provides for measuring the width of the line, determining caret location, highlighting text, and testing for a mouse hit within the text.

So why buy Postscript fonts anymore? From Apple's viewpoint, you won't have to. To quote Apple's press release "...the Apple font format will be all most Macintosh owners really need." The font format has been freely distributed to font vendors, and it has been designed so that font conversion can be done automatically. This means font vendors can easily and cheaply support the new outline font technology without having to pay royalties. Will this break the back of Adobe and Postscript by providing a cheaper alternative? Could be. Think of this: Postscript quality laser printers at half the cost of current printers, using Apple outline fonts. The current crop of official postscript printers and imagesetters (2400 dpi machines) include a license cost of upwards of \$1000 to include the Adobe Atlas RIP in each machine. That translates into a customer cost increase of \$2000. Take that away and the same \$4500 Postscript printer now costs \$2500 and gives the same results, for fonts, that is. In particular, it makes the now useless LaserWriter IISC into a very useful, and cheaper alternative to the LaserWriter NTX.

Another interesting note in the Apple press release is that while "Apple is committed to maintaining excellent system support for Postscript printing", ... "our policy is to not comment specifically on hardware products under development." Does this mean a new family of Quickdraw LaserWriter printers that support Apple Outline fonts but does not include Postscript? With the future planned enhancements to Quickdraw, will such a family of printers end up being more powerful than today's Postscript printers?

Improving Quickdraw

The obvious difference that still remains is the fact that Quickdraw is not as good as Postscript at defining a graphic image on the page. No rotated text, for example. So what is Apple doing about that? Unofficially, Apple is saying that Quickdraw is being re-written to be an object-oriented imaging system superior (and certainly faster) than Postscript. However, only 32-bit color and outline fonts will be ready in time for System 7.0. A future system release, perhaps 8.0, will include an all new Quickdraw imaging engine that will remove the final reasons for having to have Postscript at all. It will include new drawing primitives, floating point, 3D, animation, and possible hardware acceleration. The problem for Apple is, how do you extend the capability of Quickdraw, which is the heart of the Macintosh design, and not break everybody in the process? What Apple is saying, is that they will definitely *not* move to Display Postscript and that they definitely *will* upgrade Quickdraw to be at least the functional equivalent.

One area Apple could quickly move ahead in is the area of halftones and color. Postscript does not do a good job of either. In releasing print driver 6.0, Apple included the ability to half-tone a color image on the LaserWriter. If Apple can shape it's

almost fanatic interest in Multi-media presentations into the more mundane and practical problems of pre-press by improving on high-resolution typesetting, splines, halftones and color separation, they can continue to dominate an industry they created with the LaserWriter. Pre-press technology is a multi-billion dollar industry. By quickly adding halftones and color separation to low-cost imagesetters that don't include an Adobe license fee, technology that now costs \$40,000 and up can be obtained for \$20,000, providing another huge market for Macintosh computers. This is where Apple could get ahead of Postscript with it's Quickdraw enhancements and create a justifiable reason for building cheaper Quickdraw based imagesetters. At the very least, they would be much faster, something that would not be lost on the growing service industry building up around Linotronic and other high-end imagesetters. Lets hope that in improving Quickdraw, Apple puts it's energy into pre-press issues like arbitrary text on a path, rather than the more exotic but less important issues like fancy animation.

How Applications will Break

The most radical change in System 7.0 will be in the new Print Manager. All current printer drivers, including the LaserWriter, Linotronic, Compu-Graphic, Varityper and everything else out there, will be incompatible with System 7.0. Apple, will of course, have new printer drivers for Apple printer products, but what about all those expensive imagesetters out there? They will have to either be able to use Apple's LaserWriter driver, or they will have to re-write their printer driver to work with System 7.0. And applications wanting to take advantage of System 7.0 printing features will also have to make revisions to their dialogs. This one aspect of System 7.0 alone will almost guarantee every major software package vendor will be coming out with new versions to support System 7.0.

The main feature of the new printing manager is that the silly thing was re-written! It was a hopeless mess of outdated code, exceeded only by MacInTalk in ugliness. The new manager offers a consistent, uniform interface for all Macintosh computers and all printing devices, and makes printer driver writing, a matter of weeks, instead of months or even years, as it has been in the past. The intent is to allow a single image of a document on the screen (based on Quickdraw!) to go to any print type device without special casing, including a printer, another raster device, a vector device or whatever. Apple will provide "print engines" for a Quickdraw to Postscript device, Quickdraw to Raster device and Quickdraw to Vector device. Under this approach, plotters, screen capture devices and Postscript Imagesetters will all work the same way without changes to the application code.

At least, that would be the promise if Quickdraw worked the way it should. You still have the problem that Quickdraw just can't do some things that need to be done, so how do you get around that? Apple is working on trying to figure out what to do about PicComments and imbedded Postscript that many applications use to extend Quickdraw's limited imaging ability. Whatever they come up with certainly will be designed to make future system expansion easier as Quickdraw evolves. One thing is for

sure; nearly every application will have to be modified to take advantage of the new system. PrOpenPage, for example, has been changed to PrStartPage. The format of PicComments to indicate the start and end of postscript has also been changed. PrEndPage is used instead of PrClosePage. In short, everything is different. Of all the parts of System 7.0, this is the one that scares me the most. It impacts everything, and is intimately connected with Quickdraw, the new outline fonts and the Layout Manager in it's implementation. Let's hope they do it right. One advantage of the new attitude is that Apple will actually encourage vendors to license the new printing driver toolkit to create new print drivers, even if the product competes with an Apple product. This should open up the creation of new peripherals for the Mac considerably.

New Finder Stuff

The Finder has been improved greatly under System 7.0. As was mentioned, there will only be MultiFinder, on all the time. So plan on always using WaitNextEvent! The file organization on the Mac has been changed. It now uses an HFS B-Tree mechanism for improved performance. New toolbox calls are provided for applications to access this B-Tree capability. Searching an 80 meg hard disk can be done quickly in 3 seconds.

Files now have a unique FileID reference that is insensitive to either a folder move, or name change. This is designed to eliminate the infamous "lost dictionary" dialogs. A toolbox call resolves the FileID into the customary directory ID and File name.

The Desktop file has been re-designed. It now has an entire Desktop Manager to manage it! This is supposed to improve performance on large hard disks with many files. A new File System Manager has been created that allows an application to use a foreign file system; i.e., MS/DOS. This should make it easy to write Mac applications that read and manage MS/DOS disks directly. Database access has been improved by providing a built-in query language called CL/1. This allows any application to become "data base aware" for accessing information on mainframe networks.

The sound manager has been re-written to support multiple channels, sequencing, a new MIDI manager, and audio compression and expansion. Kirk Austin is trying desperately to get some software out of Apple to give us a full report on how to plan on making use of all the new sound features.

Can't Talk Yet

One area that Apple did not improve on is MacInTalk. It seems that MacInTalk was a one shot deal that Steve Jobs had done just before the grand Mac opening in 1984. He wanted Mac to "introduce itself" with speech. The guys who wrote it offered to sell the source code to Apple, but the price was too high and Apple refused. The result is that Apple doesn't have MacInTalk and never did! The fixes in version 1.31 were done by hacking the disassembled code to get it to run on a Mac II! One of the clear messages Apple got back from the conference was that people want an improved MacInTalk that is supported. Apple has

promised to provide an interim solution of a new MacInTalk that is slightly better, supported and runs about 100K, in the short term; i.e., the next twelve months.

Tool Making Is Back!

Finally, there are several user interface enhancements. Desk accessories are still supported, but now, any application can be a desk accessory by simply dropping it into a special Apple folder. So there is no reason to write a DA any more. Just write a small, dedicated application instead. This should give birth to a new tool industry on the Mac. With virtual memory, IAC and arbitrary desk accessories, small, powerful, single function tools are now the way to go in Mac software. Forget the monolithic, three years late, 900K integrated mega-application! Develop four well-connected 200K tool applications instead and drop them in the desk accessory folder! You can do one every six months and eat during your development cycle!

Early Christmas Present

Apple has begun releasing some of the System 7.0 goodies early. These include 32-Bit Quickdraw and the new LaserWriter 6.0. The following list of files give the updated versions of the color quickdraw release that are different from those supplied with System Release 6.03, the current latest version:

- | | |
|-------------------------|---------------------|
| • General: 3.3.2 | • Monitors: 4.0 |
| • 32-bit Quickdraw: 1.0 | • PrintMonitor: 1.3 |
| • LaserPrep: 6.0 | • LaserWriter: 6.0 |

If you have these versions of these files, then you have the new color quickdraw updates. They are further identified as coming from "Apple's Color Disk 1.0" in the GetInfo dialog box.

In addition to getting "true color" for the new 24-bit color video boards, you also get Color Quickdraw to color PostScript conversion and halftoning of color images on monochrome Postscript printers. And the new color quickdraw includes a BitmapToRegion function!

The Future

So what can we say about the future? Get into object-oriented programming and MacApp! The direction Apple is going clearly indicates that Mac system software is converging with object-oriented programming and it is only a matter of time before the toolbox is entirely object oriented. The fact that the new Finder was written in C++ should be indication enough of the change that has taken place within Apple. MacApp is now being supported at the highest levels inside Apple and very soon we should see MacApp object support at very nearly the same time we get MPW Pascal and C interface support for new system feature releases. The latest 3.0 final version of MPW and the 2.0 release of MacApp form the first fully functional, "finished" version of an object-oriented environment that MacTutor can encourage all developers, both Pascal and C, to get behind and support. For our part, we intend to devote a considerable portion of MacTutor editorial space to object Pascal, MacApp and C++ as the best way to develop Macintosh software that works with System 7.0 and beyond. *Let us hear your comments!*

Write me care of MacTutor and get yourself published.



Programmer's Forum

The Installer Explored



Steven Sheets
Contributing Editor
Hernon, VA
Volume 5, Number 8

The Installer and Creating Scripts

The one eternal truth about Macintosh software is that there will always be another System Software release. Every few months, swarms of Macintosh users will boot up new disks from Apple in order to update their machines to the latest revision. In the early days of Macintosh, this consisted of dragging the new System Folder from the new source disk to the target disk. Of course, any special files or resources that the user had placed in his System Folder would be deleted. As more Desk Accessories, Fonts, INITs, Printer Drivers and other non-Apple System Software packages were developed, the limitations of this method became obvious.

If a user has a heavily modified System Folder, he might not want to loose the special software he had installed. Reinstalling all the correct portions by hand (Font/DA Mover, ResEdit, etc.) would take time. In many cases, if someone else has modified his system folder for him, the user may not know how to recreate these changes on the new system software.

To solve this problem, Apple introduced the Installer program. Most people have seen the Installer; it is included in all Macintosh System Software releases. By booting off the update disk (source) and running the Installer program, the user's boot disks (target) can be updated. This update can be done file by file or resource by resource. Thus any special software that was on the target disks will not be disturbed. Fonts and Desk Accessories will not have to be copied over with Font/DA Mover. Special INITs will not have to be reinstalled.

The Installer program is being used by non-Apple developers as a method to install and update System Software they have written. With the Installer and custom made Scripts, Software Drivers that run special hardware are simple to install. If there are a number of Drivers, depending on what CPU the user has, multiple Scripts can be created. Each Script would only move the needed files and resources to the Target disk's System Folder. For example, in order to run a special INIT, an older CPU might require a patch that the new CPUs already have in ROM. While the vast majority of MacTutor readers are capable of modifying the System File by hand, most Mac users do not have the knowledge to do this. For example, by using the Installer and special Scripts, all the various Finder hacks can be safely done to a user's machine without ResEdit.

Scripts

The Installer program reads it's instructions from Script resources. These Scripts tell the Installer program what files and resource to copy from the Source disk's System Folder to the Target disk's System Folder. Script resources are of type 'insc'. These Script resource must be in very specific spots in order for

the Installer program to find them.

1. The file that contains the Script must be in the System Folder, the same folder as the Installer program, or any subfolder that is in the same folder as the Installer program.

2. Only one Script can be in any file in order for the Installer program to use it. As long as there is only one 'insc' resource, the ID of that resource is not important. While most Script uses ID 0, any one will work.

3. The Script resource must be inside a file that has a specific file type. Usually they are in files of type 'uins'. A file that has the owner and creator of 'uins' will have the Installer document icon (ie. double clicking on that document will execute the Installer program). Such files are usually inside a folder called 'Installer Scripts' on the update disks from Apple. The Script resource can also be inside a file of 'PRES', 'PRER' or 'RDEV' (various Printer drivers and other Chooser devices). Thus if such files are in the System Folder of the Source disk, they can contain the instruction of how to install themselves onto the Target Disk. The Laserwriter Driver uses this method.

The Script resource consists of three parts; the header information, the file list information and the resource file list information. Each part contains various data structures. Most data structures contain instructions for the Installer that the programmer must set. Other data structures are for internal use by the Installer and should be set to zero. A PStr is an even padded Pascal String (ie. single byte count N followed by N number of ASCII bytes of char). A WStr is a even padded word counted string (ie. Integer count N followed by N number of ASCII bytes of char). A Word is an integer of two bytes or 16 bits, while LongInt is an integer of 4 bytes or 32 bits. A ResType is also 4 bytes of data, but they are referenced as 4 ASCII characters, not an integer. The data structures are usually variable length; the size of a structure can vary according to the content (number of items in list and size of strings). Knowing this, a Pascal Type declaration describing the Script would look something like this:

```
TYPE
ScriptRec = RECORD
  Header: HeaderRec;
  FileList: FileListRec;
  ResFileList: ResFileListRec;
END;
```

```
HeaderRec = RECORD
  Format: INTEGER;
  Flags: INTEGER;
  Name: PStr;
  Help: WStr;
END;
```

```

FileListRec = RECORD
  NumFiles: INTEGER;
  Files: ARRAY[0..0] OF FileSpecRec;
END;

FileSpecRec = RECORD
  Flags: INTEGER;
  Typ: ResType;
  Creator: ResType;
  Date: LongInt;
  Hand: LongInt;
  FDelSize: LongInt;
  FAddSize: LongInt;
  FileName: PStr;
END;

ResFileListRec = RECORD
  NumResFiles: INTEGER;
  ResFiles: ARRAY[0..0] OF ResFileRec;
END;

ResFileRec = RECORD
  TargetFile: FileSpecRec;
  SourceFileList: SourceFileListRec;
END;

SourceFileListRec = RECORD
  NumSourceFiles: INTEGER;
  SourceFile: FileSpecRec;
  ResLists: ARRAY[0..0] OF ResListRec;
END;

ResListRec = RECORD
  NumResources: INTEGER;
  Resources: ARRAY[0..0] OF ResSpecRec;
END;

ResSpecRec = RECORD
  Flags: INTEGER;
  Typ: ResType;
  SourceID: INTEGER;
  TargetID: INTEGER;
  CRCVer: INTEGER;
  Filler1: INTEGER;
  Filler2: LongInt;
  RDelSize: LongInt;
  RAddSize: LongInt;
  Name: PStr;
  PrevCRC: INTEGER;
END;

```

Remember that these Type declarations are not legal, but they are good tools to use to visualize the data structures.

Header Information

The header information contains the Script's Format, Flags, Name and Help Message. The Format and Flags are internal integers and should be set to zero. The Name is a PStr containing the name as it will appear in the Installer program. The Name should include a revision number of the Script. The Help Message is WStr that will appear when the user presses the Help button in the Installer program.

File List Information

The File List contains the instruction about which files are to be manipulated (copied, deleted) from the Source Disk to the Target Disk. The File List starts with an integer number,

followed by a variable length array of File Specs. The integer is the number of elements of the array. Each element of the array contains information about each File that need to be manipulated.

The File Spec data structure is used in the File List and the Resource List. It is used to pick a specific file and describe actions done to that file. A File Spec contains the following fields: Flags, Type ID, Owner ID, Date, Handle, FDelSize, FAddSize and Name. The Handle, FDelSize, and FAddSize are LongInts that are used internally and should always be set to Zero. If the Type ID (ResType), Owner ID (ResType) and Date (LongInt) are used, they should match the Type, Owner and Date of the Target file. If they are not used, they should be set to Zero. The Name is a PStr, matching the Target files name. The Flag is a 2 byte value (16 bit) where most of the bits are Boolean flags. It is the Flag bits that actual tell the program what to do with the file. The following is their function (in least significant order).

Bit Name Function

0 — Set to One

- 1 TypeCr: If set, the Source & Target File's Type and Creator must match the Date Field. If it does not match, the update will not be attempted. If unset, the Type and Creator are ignored.
- 2 CrDate: If set, the Source File's Creation date must match the Date Field. If it does match, the update will not be attempted. If unset, the Date will be ignored.
- 3 needExis:t Source File needs to exist. If it doesn't, the update will not be attempted. If unset, the Source file does not need to exist, and the Script will continue.
- 4 DataFork: If set, when the Source file is copied to the Target File, the Data Fork of the Source file will be copied. If unset, the Data fork will not be copied.
- 5 RsrcFork: If set, when the Source file is copied to the Target File, the Resource Fork of the Source file will be copied. If unset, the Resource fork will not be copied.
- 6 UpdateOnly: If set and the Target file does not already exist, then the Source file will not be copied (regardless if Bit 1 is set). In that case, the Script will continue.
- 7 Old: If set, even if TypeCR is set, Target File's Type & Creator need not match. The Source File's Type & Creator need still need to match if TypeCR is set.
- 8 — Set to Zero
- 9 Replace: Sorry folks, there is no documentation to explain this bit. However all the examples and sample have it turned on, and they work. So always set it!
- 10 — Set to Zero
- 11 — Set to Zero
- 12 — Set to Zero
- 13 Copy: If set, when the user clicks the Install button, the Source File will be copied to the Target File.

- 14 DelInstall: If set and a Target file already exists, then that Target file will be deleted before the new Target File is copied from the Source Disk.
- 15 DelRemove: If set and a Target file exists, when the user clicks the Remove button, the Target file will be deleted.

Resource File List Information

The Resource File List contains instructions about which resources of which files are to be manipulated (copied, deleted) from the Source Disk to the Target Disk. The Resource File List starts with an integer number, followed by a variable length array. The integer is the number of elements of the array. Each element of the array contains a Target File Spec and a Source File List. The Target File Spec describes which file to copy the resources to. The Source File List describes, file by file, the resources to be moved. It contains a Source File Spec (where to get the resources) and a Resource List. Once again, the Resource List starts with an integer number, followed by a variable length array. Each element of the array is a Resource Spec.

The Resource Spec contains the information describing exactly which resource to move. The Resource Spec contains the following fields: Flags, Type, Source ID, Target ID, CRC Version, Filer1, Filer2, RDelSize, RAddSize, Resource Name and Previous CRC List. Again, CRC Version, Filer1, Filer2, RDelSize, RAddSize, and Previous CRC List are all used internally and should be set to zero.

When a resource is moved from the Target disk to the Source disk, it does not need to have the same resource ID as it started out with. The Source ID is the resource number of the resource on the Source Disk. The Target ID is the resource number of the resource when it is moved to the Target Disk. In many cases, the number are the same, but they do not always have to be. The Type field is the resource Type of the moved resource on the Source Disk and the Target Disk. Thus when a resource is moved, it can not change types like it does IDs. The Resource Name is a PStr. It's usage is dependent on the Flags. Again, the Flags field is a group of Boolean Bit flags, whose usage is defined as:

Bit Name Function

- 0 — Set to Zero
- 1 ByID: If set, find the source resource by using the Source ID. If unset, find the source resource by using the Resource Name.
- 2 — Set to Zero
- 3 needExist: Source Resource needs to exist. If it doesn't, the update will not be attempted. If unset, the Source Resource does not need to exist, and the Script will continue.
- 4 EvenIfProt: If set & the Target Resource already exists, protecting the resource will not prevent a new Target Resource from being copied over it. Nor will it protect it from a deletion. If unset, and the Target is protected, the original Target Resource can not be deleted or copied over. In

that case, an update will not be attempted.

- 5 — Set to Zero
- 6 UpdateOnly: If set and the Target Resource does not exist, then the Source file will not be copied (regardless if Bit 1 is set). In that case, the Script will continue.
- 7 OldName: If set, and if ByID is set, Target Resource's Name need not match the Resource Name.
- 8 — Set to Zero
- 9 — Set to Zero
- 10 — Set to Zero
- 11 — Set to Zero
- 12 — Set to Zero
- 13 Copy: If set, when the user clicks the Install button, the Resource in the Source File will be copied to the Target File.
- 14 DelInstall: If set, and if a Target Resource already exists, then that resource is deleted before the new Target Resource is copied from the Source Disk. For most cases, this bit must be set, or else, when the Target Resource is copied over, two resources with the same number and type will exist in one file. Somehow the file's resource map becomes corrupted.
- 15 DelRemove: If set and if a Target resource exists, when the User clicks the Remove button, the Target Resource will be deleted.

Tools

The best tool in which to create an Installer Script is MPW's Rez. While any Resource editor (ResEdit, RMaker, etc.) can create a Script, it is easiest to create one while using Rez. The various Flag bits are manipulated as Boolean values and do not have to be calculated into an Integer. All of the preset data structures (empty handles and bits) are done by Rez, and are invisible to the user. List Counts are calculated by Rez, as are word alignments.

Rez and Derez use Include files to define the format of various resources. The format of a 'inisc' type resource is contained in the MPW file "SysTypes.r". Any use of Rez or Derez must reference this file.

Examples

In many cases, a Script file contains only the instructions on how to move the resources and files from one disk's System Folder to another. Those types of Script files can be in the System Folder or in the folder that contains the Installer program or in some subfolder of the folder that contains the Installer program. However, while many Script files contain only the instructions, other Script files can contain the resources that are to be moved. In these cases, the Script files must be in the Source disk's System Folder. The Installer program can thus access the file for both instructions and resources. The following resource code examples create Script files of that type. They are compiled using the MPW Rez tool.

FlameTrash and NormalTrash are example of Scripts that

modify existing System Software. In both cases, the Script is designed to copy the Trash Can ICN# resources. The Flags are set so that resource numbers 500 & 501 in the Script file are copied to resource numbers 130 & 134 in the Finder file. The various files are found by their Name, Creator & Type. Since the Script was not meant for a specific revision of the Finder, the CrDate flag was unset and the Date field was not used. The resources are found by resource ID & type, so the resource name is not important. These Scripts have no remove feature since the Finder must always have the Trash Can resources. Thus to change the Trash Can to normal, the NormalTrash Script must be used.

SampleProc is an example of how to install new resources and files into the System Folder. This type of Script can be used to install new Drivers, Fonts, or various Macintosh Procedure code resources like MenuProc, WindowProc, ControlProc, etc. SampleProc copies a dummy resource (number 0 and type 'XXXX') from the Script into the System file. The Script also copies itself into the System Folder. Thus if the Target disk has the Installer program on it, it is ready to install the SampleProc on some other disk. The same technique can be used to install other files; as an example, the Laserwriter Script installs the Laserwriter Prep file. In this Script, the DelRemove flag is set for the resource and the file, since both can be deleted without any problems. By pressing the Remove Button, the new SampleProc script is deleted and the 'XXXX' resource is removed from the System File. Even though the Script only has a Data Fork, the Flags DataFork and RsrcFork must both be set in the File List for the file to be completely deleted. Only setting one Flag means that only one fork is deleted; the file still exists in the directory.

Listing: FlameTrash.r

```
/*
 * Flame TrashCan Installer Script Resource Code
 * Create using with command:
 *   Rez -o FlameTrash -t uins -c uins FlameTrash.r
 * To Function correctly, Place it in the System Folder of the
 *   Installer Disk.
 * Installs the Flame TrashCan into the Target Disk's Finder.
 */

#include "Types.r"
#include "SysTypes.r"

resource 'insc' (0, "") (
    format0 (
        "Flame TrashCan (v1.0)",
        "FlameTrash Script installs the Flame Trashcan into the
        Finder. "
        "This Script must be in the System Folder of the Install-
        ing Disk. "
        "It was created for MacTutor by Steve Sheets.",
        ( /* array FileList: 0 elements */
        ),
        ( /* array ResFileList: 1 elements */
        /* [1] */
        noDelRemove, noDelInstall, noCopy, doReplace, Old, noUp-
        dateOnly,
        noRsrcFork, noDataFork, needExist, noCrDate, typeCr,
        'FNDR', 'MACS', 0x0, "Finder",
        ( /* array SrcFileList: 1 elements */
        /* [1] */
        noDelRemove, noDelInstall, noCopy, doReplace, Old,
        noUpdateOnly,
```

```
noRsrcFork, noDataFork, needExist, noCrDate, typeCr,
'uins', 'uins', 0x0, "FlameTrash",
( /* array ResList: 2 elements */
/* [1] */
noDelRemove, DelInstall, copy, doReplace, old,
noUpdateOnly,
noEvenIf, needExist, byID,
'ICN#', 500, 130, "",
/* [2] */
noDelRemove, DelInstall, copy, doReplace, old,
noUpdateOnly,
noEvenIf, needExist, byID,
'ICN#', 501, 134, ""
)
)
)
);
/*Flame Trash Icon */
resource 'ICN#' (500, "", preload) (
    ( /* array: 2 elements */
    /* [1] */
    $*1F FF FF F8 20 00 00 04 40 00 00 02 80 00 00 01"
    $*80 00 00 01 80 00 00 01 80 20 00 01 80 18 00 01"
    $*80 0C 00 01 80 06 00 01 80 07 00 01 80 03 80 01"
    $*80 06 C0 01 80 06 60 01 80 0C 30 01 80 18 18 01"
    $*80 30 0C 01 80 61 06 01 80 C1 83 01 80 81 41 01"
    $*81 82 21 81 81 84 11 81 81 84 91 81 80 C4 93 01"
    $*80 E2 A7 01 80 71 CE 01 80 39 9C 01 80 1F F8 01"
    $*80 07 E0 01 40 01 80 02 20 01 80 04 1F FF FF F8",
    /* [2] */
    $*1F FF FF F8 3F FF FF FC 7F FF FF FE FF FF FF FF"
    $*FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF"
    $*FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF"
    $*FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF"
    $*FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF"
    $*FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF"
    $*FF FF FF FF 7F FF FF FE 3F FF FF FC 1F FF FF F8"
    )
    );
/*Flame Full Trash Icon */
resource 'ICN#' (501, "", preload) (
    ( /* array: 2 elements */
    /* [1] */
    $*1F FF FF F8 20 40 00 04 40 20 40 02 80 38 20 01"
    $*80 1C 30 01 84 16 38 01 83 1B 3C 01 81 89 B6 01"
    $*81 8C E7 01 81 EC 63 81 83 74 31 81 83 34 18 C1"
    $*86 1C 18 61 86 18 0C 61 8C 30 06 71 8C 60 06 31"
    $*98 60 03 19 B8 C0 03 19 B0 C0 03 19 B0 C0 06 19"
    $*B8 60 86 39 9C 70 8C 71 8E 38 9C E1 87 18 B9 C1"
    $*83 8E B7 81 81 C7 EF 01 80 71 DE 01 80 3F FC 01"
    $*80 0F F0 01 40 01 80 02 20 01 80 04 1F FF FF F8",
    /* [2] */
    $*1F FF FF F8 3F FF FF FC 7F FF FF FE FF FF FF FF"
    $*FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF"
    $*FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF"
    $*FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF"
    $*FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF"
    $*FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF"
    $*FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF"
    $*FF FF FF FF 7F FF FF FE 3F FF FF FC 1F FF FF F8"
    )
    );
);
```

Listing: NormalTrash.r

```
/*
 * Normal TrashCan Installer Script Resource Code
 * Create using with command:
 *   Rez -o NormalTrash -t uins -c uins NormalTrash.r
 * To Function correctly, Place it in the System Folder of the
 *   Installer Disk.
 * Installs the Normal TrashCan into the Target Disk's Finder.
 */
```

```

#include "Types.r"
#include "SysTypes.r"

resource 'insc' (0, "") (
    format0 (
        "Normal TrashCan (v1.0)",
        "NormalTrash Script installs the Normal Trashcan into the
        Finder. "
        "This Script must be in the System Folder of the Install-
        ing Disk. "
        "It was created for MacTutor by Steve Sheets.",
        ( /* array FileList: 0 elements */
        ),
        ( /* array ResFileList: 1 elements */
        /* [1] */
        noDelRemove, noDelInstall, noCopy, doReplace, Old,
        noUpdateOnly,
        noRsrcFork, noDataFork, needExist, noCrDate, typeCr,
        'FNDR', 'MACS', 0x0, "Finder",
        ( /* array SrcFileList: 1 elements */
        /* [1] */
        noDelRemove, noDelInstall, noCopy, doReplace, Old,
        noUpdateOnly,
        noRsrcFork, noDataFork, needExist, noCrDate, typeCr,
        'uins', 'uins', 0x0, "NormalTrash",
        ( /* array ResList: 2 elements */
        /* [1] */
        noDelRemove, DelInstall, copy, doReplace, old,
        noUpdateOnly,
        noEvenIf, needExist, byID,
        'ICN#', 500, 130, "",
        /* [2] */
        noDelRemove, DelInstall, copy, doReplace, old,
        noUpdateOnly,
        noEvenIf, needExist, byID,
        'ICN#', 501, 134, ""
        )
        )
    )
);

/*Normal Trash Icon*/
resource 'ICN#' (500, "", preload) (
    ( /* array: 2 elements */
    /* [1] */
    $00 07 E0 00 00 08 10 00 03 FF FF C0 04 00 00 20"
    $07 FF FF E0 02 00 00 40 02 00 00 40 02 44 44 40"
    $02 22 22 40 02 22 22 40 02 22 22 40 02 22 22 40"
    $02 22 22 40 02 22 22 40 02 22 22 40 02 22 22 40"
    $02 22 22 40 02 22 22 40 02 22 22 40 02 22 22 40"
    $02 22 22 40 02 22 22 40 02 22 22 40 02 22 22 40"
    $02 22 22 40 02 22 22 40 02 22 22 40 02 22 22 40"
    $02 44 44 40 02 00 00 40 02 00 00 40 03 FF FF C0",
    /* [2] */
    $00 07 E0 00 00 0F F0 00 03 FF FF C0 07 FF FF E0"
    $07 FF FF E0 03 FF FF C0 03 FF FF C0 03 FF FF C0"
    $03 FF FF C0 03 FF FF C0 03 FF FF C0 03 FF FF C0"
    $03 FF FF C0 03 FF FF C0 03 FF FF C0 03 FF FF C0"
    $03 FF FF C0 03 FF FF C0 03 FF FF C0 03 FF FF C0"
    $03 FF FF C0 03 FF FF C0 03 FF FF C0 03 FF FF C0"
    $03 FF FF C0 03 FF FF C0 03 FF FF C0 03 FF FF C0"
    )
);

/*Normal Full Trash Icon*/
resource 'ICN#' (501, "", preload) (
    ( /* array: 2 elements */
    /* [1] */
    $00 07 E0 00 00 08 10 00 03 FF FF C0 04 00 00 20"
    $03 FF FF C0 02 00 00 40 02 00 00 40 02 10 04 40"
    $04 22 22 20 04 22 22 20 08 44 11 10 08 44 11 10"
    $10 88 08 88 10 88 08 88 10 88 08 88 10 88 08 88"
    $10 88 08 88 10 88 08 88 10 88 08 88 10 88 08 88"
    $10 88 08 88 10 88 08 88 10 88 08 88 10 88 08 88"
    )
);

```

```

$08 44 11 10 04 22 22 20 04 22 22 20 04 22 22 20"
$02 10 00 40 02 00 00 40 02 00 00 40 01 FF FF 80",
/* [2] */
$00 07 E0 00 00 0F F0 00 03 FF FF C0 07 FF FF E0"
$03 FF FF C0 03 FF FF C0 03 FF FF C0 03 FF FF C0"
$07 FF FF E0 07 FF FF E0 0F FF FF F0 0F FF FF F0"
$1F FF FF F8 1F FF FF F8 1F FF FF F8 1F FF FF F8"
$1F FF FF F8 1F FF FF F8 1F FF FF F8 1F FF FF F8"
$1F FF FF F8 1F FF FF F8 1F FF FF F8 1F FF FF F8"
$0F FF FF F0 07 FF FF E0 07 FF FF E0 07 FF FF E0"
$03 FF FF C0 03 FF FF C0 03 FF FF C0 01 FF FF 80"
);

```

Listing: SampleProc.r

```

/*
 * SampleProc Installer Script Resource Code
 * Create using with command:
 * Rez -o SampleProc -t uins -c uins SampleProc.r
 * To Function correctly, Place it in the System Folder
 * of the Installer Disk
 * Installs a Sample Resource into the Target Disk's
 * System File. Also it copies the Script into the
 * Target Disk's System Folder so it can Install
 * itself elsewhere. This Script could be used to
 * install almost any type of Proc Resource, for
 * example, WindowProc, MenuProc, ControlProc, etc.
 */

#include "Types.r"
#include "SysTypes.r"
resource 'insc' (0, "") (
    format0 (
        "SampleProc (v1.0)",
        "For the SampleProc Script, pressing the Install Button "
        "installs the SampleProc Resource (Type 'XXXX', ID 0) "
        "into the System File. It also copies the SampleProc "
        "Script into the System Folder. Pressing the Remove
        Button, "
        "deletes the Resource and the Script. This Script must "
        "be in the System Folder of the Installing Disk. It "
        "was created for MacTutor by Steve Sheets",
        ( /* array FileList: 1 elements */
        /* [1] */
        DelRemove, DelInstall, Copy, doReplace, Old, noUp-
        dateOnly,
        RsrcFork, DataFork, needExist, noCrDate, typeCr,
        'uins', 'uins', 0x0, "SampleProc"
        ),
        ( /* array ResFileList: 1 elements */
        /* [1] */
        noDelRemove, noDelInstall, noCopy, doReplace, Old,
        noUpdateOnly,
        noRsrcFork, noDataFork, needExist, noCrDate, typeCr,
        'ZSYS', 'MACS', 0x0, "System",
        ( /* array SrcFileList: 1 elements */
        /* [1] */
        noDelRemove, noDelInstall, noCopy, doReplace, Old,
        noUpdateOnly,
        noRsrcFork, noDataFork, needExist, noCrDate, typeCr,
        'uins', 'uins', 0x0, "SampleProc",
        ( /* array ResList: 1 elements */
        /* [1] */
        DelRemove, DelInstall, copy, doReplace, old,
        noUpdateOnly,
        noEvenIf, needExist, byID,
        'XXXX', 0, 0, ""
        )
        )
    )
);

/*SampleProc Resource */
data 'XXXX' (0) (
);

```



Jörg's Folder

Compiler Comparisons



Jörg Langowski
MacTutor Editorial Board
Volume 5, Number 8

Code optimization

You will have noticed the change in the column's title: a recent reader survey has shown that Forth and Basic are the two languages that our readers would most like to see *less* of in MacTutor. That's a shame, but we're responsive (at least we try)...

So, from now on my monthly column will have a wider scope. As you might have seen, I have very often used Forth as a vehicle to explain general concepts of Macintosh programming. Since many subscribers don't seem to be happy with Forth - in fact, people often have asked about things that had been explained in a Forth column, but they just hadn't read. The column about the Notification Manager in V5#6 is a good example: at the bottom of page 42, one of the 'ideas to be written' was explained as: "*An example that places a Notification Manager request in low System memory, and starts a Timer routine...*"; this is just the example that was in the Forth Forum that covered the Notification Manager. Anyway, I'll try to use other vehicles to convey the message from now on. Such as assembly, or maybe even C. Mach2 Forth still is a very good assembly-language development system because of its interactivity. Of course, you cannot create complicated macros, or structures, and have to resort to Forth code for those purposes. I'll still inform you about interesting things I come across on the Forth scene, but this won't be an exclusive Forth column anymore. Emphasis will be on two things: basic system-level things such as drivers, trap patches, INITs, network, new managers as they come up; and - on the other side of the spectrum - object-oriented programming in C++.

Apple will 'Real Soon Now' release a C++ under MPW, and we'll hopefully have a pre-release by the time you read this. C++ is a very interesting language, much more than a simple extension of C; reading Stroustrup's book I got this feeling of 'yes, that's how one should have done it in the first place' that I had 15 years ago when all I knew was Algol 60, and came across the description of Algol 68. Sadly enough, Algol 68 never really caught on; hopefully, C++ will. The C++ column will start with the next issue; including program examples if we get the pre-release soon enough, just 'dry swimming' if not.

This month, we'll talk once more about one of my favorite subjects, number crunching, speed (or the lack of it), and intelligence in compilers (or the lack of it).

A matrix multiplication routine

Since I am doing more and more everyday computation (mostly Fortran) on the MacII, I'm obviously interested in a good optimizing compiler. Now, a standard trick that every decent compiler should have in its repertoire is the elimination of

constant expressions from loops, or assignment of array elements that are not dependent on the loop index to intermediate variables.

Imagine my surprise when I found out that (in Language Systems Fortran 1.2.1) I could speed up a loop that looked like this:

```
do k=1,n3
  do i=1,n1
    c(i,k) = 0x0
    do j=1,n2
      c(i,k) = c(i,k)+a(i,j)*b(j,k)
    end do
  end do
end do
```

by simply writing:

```
do k=1,n3
  do i=1,n1
    sum = 0x0
    do j=1,n2
      sum = sum+a(i,j)*b(j,k)
    end do
    c(i,k) = sum
  end do
end do
```

Now, in undergraduate programming classes, years ago, we were actually taught to look for constant arithmetical or index expressions in loops and put them outside if possible. Today, almost everybody assumes that the compiler is smart enough to take care of that; incorrectly, as you see. To see how good the compilers available under MPW can do, I wrote a Fortran program (listing 1) that calls several versions of this matrix multiplication loop, written in Fortran (Lang. Sys. 1.2.1), Pascal (Apple 3.0 beta), and C (Apple 3.0 beta). Surprise: *none* of the compilers was good enough to move the indexing outside of the loop. The following table gives the results (Mac IIx):

Pascal, hand-optimized:	2.7667 seconds
C, register variables, hand-opt.:	3.4667 seconds
Pascal:	4.0333 seconds
Fortran, const. dimensions, opt=3:	4.5000 seconds
Fortran, hand-optimized, opt=3:	4.6500 seconds
C:	4.7167 seconds
Fortran, opt=3:	6.5167 seconds
Fortran, opt=0:	6.6167 seconds

A difference of more than a factor of 2 between the slowest Fortran and the fastest Pascal code. Apple Pascal lived up to its good reputation here, but even that could be improved a lot by

eliminating the constant index expression.

Surprised, I ran the Fortran benchmark on a Microvax II, and found that even there some speed could be gained by 'hand-optimizing' the code:

Fortran, plain: 3.6333 seconds
Fortran, hand-opt.: 3.2833 seconds
Fortran, const. dimensions: 3.1000 seconds

However, the difference between the machine-optimized and the hand-optimized version is not quite as big as for the MPW languages (15% for the VAX vs. 27-30% for MPW). If you compile the VAX code without optimization, you get a bigger difference (23%):

Fortran, plain: 6.2500 seconds
Fortran, hand-opt.: 4.8833 seconds
Fortran, const. dimensions: 4.8000 seconds

Therefore, take-home lesson one: don't take compiler optimizations for granted.

The machine code behind it

Benchmarks have been run on lots of different machines, using lots of different compilers. I was interested in how the code generated by the MPW compilers actually differed. A job for Nosy, and the results are shown in the last listing. I've only printed the innermost loops. Don't be overwhelmed by the pile of assembly code, just note some important details.

First, for the loop optimization examples discussed here, there seems to be no tradeoff between code length and speed. On the contrary, the fastest code is also the shortest. On the other hand, there are some obvious pieces of code which are clearly redundant. The most blatant example is the Fortran-generated code at the end of the listing, where an index expression is recalculated that was actually in register A1 all the time! 14 extra lines of machine code on each pass through the loop will add up to quite some extra time lost. Another point is that Language System obviously has no great trust in the quality of the 68000/20/30, otherwise how can one explain that they repeat the EXT.L D2 instruction each time it occurs? To make sure it works at least once?

Language Systems Fortran makes other funny assumptions about the machine, for instance it seems to think there are only two floating point registers in the 68881, FP0 and FP7. I have looked at some code which had great potential for optimization by using enough FP registers. Language Systems is, however, known for its responsiveness towards customers, so I hope we won't have to wait too long until a well-optimized Fortran shows up.

Both Pascal and C like juggling floating point registers. Why generate (like Apple's C):

```
FMOVE  FP7,FP1
FADD   FP0,FP1
FMOVE  FP1,FP7
when a simple FADD FP0,FP7 would suffice? Eliminates
```

two floating point instructions per loop. Pascal does

```
FADD   FP7,FP0
```

```
FMOVE  FP0,FP7
```

when a simple inversion of the operands

```
FADD   FP0,FP7
```

would give the same result. One floating point instruction per loop eliminated. The timing difference between the Pascal and C routines is partly because of the one extra floating point instruction.

Last remark: I haven't seen the Absoft MPW 3.0 Fortran yet. If anyone from Absoft is reading this, I'd like an evaluation copy to run the same analysis (since you claim in your ads you have such a great optimizer). If I get enough other languages collected together, we'll have a follow-up on this article.

Next month

The MacHack is over (thanks, Aimée, Carol, and all the others, for organizing such a good meeting), and I'll tell you some of my impressions in the next column. Otherwise, we'll start with an introduction to C++; I hope the compiler will arrive here in time.

Listing 1: Matrix multiplication benchmark

```
!!S Main
  program matrix
c
c Main program in Language Systems Fortran
c
c Some line breaks in the Fortran program are due to
c editing.
c
  implicit none

  integer i,j,ticks1,ticks2
  extended a(50,50), b(50,50), c(50,50)
  extended time1,time2

  integer ticks

  type *, 'Matrix multiplication benchmark'
  type *, '_____ '
  type *
  type *, 'This program compares the number crunching power'
  type *, 'of some of the popular MPW compilers.'
  type *, 'Written under MPW 3.0 by J. Langowski / MacTutor'
1989'
  type *
  type *, 'Setting up 50x50 matrices...'

  ticks1 = ticks()

  do i=1,50
    do j=1,50
      a(i,j) = (i-1) + j-1
      b(j,i) = a(i,j)
    end do
  end do

  ticks2 = ticks()
  time1 = (ticks2-ticks1)/60.
  type *
  write (6, '(f8.4, '' seconds for setting up matrices'')')
time1

  ticks1 = ticks()
  call mat_mult_for3(c,50,a,50,b,50,50,50,50)
```

```

ticks2 = ticks()
time1 = (ticks2-ticks1)/60.
type *
write (6, '(f8.4, '' seconds for multiplying matrices'',
* '' using FORTRAN routine, opt=3''') time1
type *, 'c(25,25) = ', c(25,25)

ticks1 = ticks()
call mat_mult_for(c,50,a,50,b,50,50,50,50)

ticks2 = ticks()
time1 = (ticks2-ticks1)/60.
type *
write (6, '(f8.4, '' seconds for multiplying matrices'',
* '' using FORTRAN routine, opt=0''') time1
type *, 'c(25,25) = ', c(25,25)

ticks1 = ticks()
call mat_mult_for1(c,50,a,50,b,50,50,50,50)

ticks2 = ticks()
time1 = (ticks2-ticks1)/60.
type *
write (6, '(f8.4, '' seconds for multiplying matrices'',
* '' using FORTRAN routine, hand-optimized''') time1
type *, 'c(25,25) = ', c(25,25)

ticks1 = ticks()
call mat_mult_for0(c,50,a,50,b,50,50,50,50)

ticks2 = ticks()
time1 = (ticks2-ticks1)/60.
type *
write (6, '(f8.4, '' seconds for multiplying matrices'',
* '' using FORTRAN routine, constant dimensions''') time1
type *, 'c(25,25) = ', c(25,25)

ticks1 = ticks()
call
mat_mul_pas(c,%val(50),a,%val(50),b,%val(50),%val(50),%val(50),%val(50))

ticks2 = ticks()
time1 = (ticks2-ticks1)/60.
type *
write (6, '(f8.4, '' seconds for multiplying matrices'',
* '' using PASCAL routine''') time1
type *, 'c(25,25) = ', c(25,25)

ticks1 = ticks()
call
mat_mul_pas_opt(c,%val(50),a,%val(50),b,%val(50),%val(50),%val(50),%val(50))

ticks2 = ticks()
time1 = (ticks2-ticks1)/60.
type *
write (6, '(f8.4, '' seconds for multiplying matrices'',
* '' using PASCAL routine, hand-optimized''') time1
type *, 'c(25,25) = ', c(25,25)

ticks1 = ticks()
call
mat_mul_c(c,%val(50),a,%val(50),b,%val(50),%val(50),%val(50),%val(50))

ticks2 = ticks()
time1 = (ticks2-ticks1)/60.
type *
write (6, '(f8.4, '' seconds for multiplying matrices'',
* '' using C routine''') time1
type *, 'c(25,25) = ', c(25,25)

ticks1 = ticks()
call
mat_mul_c_opt(c,%val(50),a,%val(50),b,%val(50),%val(50),%val(50),%val(50))

```

```

ticks2 = ticks()
time1 = (ticks2-ticks1)/60.
type *
write (6, '(f8.4, '' seconds for multiplying matrices'',
* '' using C routine, hand-optimized''') time1
type *, 'c(25,25) = ', c(25,25)

end

!!S Main
subroutine mat_mult_for3(c,nc,a,na,b,nb,n1,n2,n3)
c sets c=a.b
c na,nb,nc are first dimensions
c n1 n2 n3 are problem dimensions
c c is n1xn3
c a n1 n2
c b n2 n3
implicit none

integer na,nb,nc,n1,n2,n3
integer*2 i,j,k
extended c(nc,n3),a(na,n2),b(nb,n3)

do k=1,n3
do i=1,n1
c(i,k) = 0x0
do j=1,n2
c(i,k) = c(i,k)+a(i,j)*b(j,k)
end do
end do
end do
return
end

!!S Main
subroutine mat_mult_for1(c,nc,a,na,b,nb,n1,n2,n3)
c
c same as before, invariant matrix element eliminated from
loop
c
implicit none

integer na,nb,nc,n1,n2,n3
integer*2 i,j,k
extended c(nc,n3),a(na,n2),b(nb,n3)
extended sum

do k=1,n3
do i=1,n1
sum = 0x0
do j=1,n2
sum = sum+a(i,j)*b(j,k)
end do
c(i,k) = sum
end do
end do
return
end

!!S Main
subroutine mat_mult_for0(c,nc,a,na,b,nb,n1,n2,n3)
c
c same as before, with constant dimensions
c
implicit none

integer na,nb,nc,n1,n2,n3
integer*2 i,j,k
extended c(50,50),a(50,50),b(50,50)
extended sum

do k=1,n3
do i=1,n1
sum = 0x0

```

```

        do j=1,n2
            sum = sum+a(i,j)*b(j,k)
        end do
        c(i,k) = sum
    end do
end do
return
end

!!S Main
integer function ticks
ticks = long(362)
return
end

```

Listing 2 : non-optimized Fortran routine

```

!!S Main
subroutine mat_mult_for(c,nc,na,nb,n1,n2,n3)
c
c duplicate of mat_mul_for3 for compiling without optimization
c
implicit none

integer na,nb,nc,n1,n2,n3
integer*2 i,j,k
extended c(nc,n3),a(na,n2),b(nb,n3)

do k=1,n3
    do i=1,n1
        c(i,k) = 0x0
        do j=1,n2
            c(i,k) = c(i,k)+a(i,j)*b(j,k)
        end do
    end do
end do
return
end

```

Listing 3 : Pascal routine

```

($S Main)
($R-)
unit matmul;

interface

type matrix = array [1..50,1..50] of extended;

procedure mat_mul_pas
(var c : matrix; nc : longint;
 var a : matrix; na : longint;
 var b : matrix; nb : longint;
 n1,n2,n3:longint);

procedure mat_mul_pas_opt
(var c : matrix; nc : longint;
 var a : matrix; na : longint;
 var b : matrix; nb : longint;
 n1,n2,n3:longint);

implementation

procedure mat_mul_pas;
var
i,j,k:integer;
begin
for k:=1 to n3 do
for i:=1 to n1 do
begin
c[i,k] := 0;
for j:=1 to n2 do
c[i,k] := c[i,k]+a[i,j]*b[j,k];
end;
end;
end;
end;

```

```

procedure mat_mul_pas_opt;
var
i,j,k:integer; sum:extended;
begin
for k:=1 to n3 do
for i:=1 to n1 do
begin
sum := 0;
for j:=1 to n2 do
sum := sum+a[i,j]*b[j,k];
c[i,k] := sum;
end;
end;
end;
end.

```

Listing 4 : C routine

```

pascal void mat_mul_c
(extended c[50][], long nc,
 extended a[50][], long na,
 extended b[50][], long nb,
 long n1, long n2, long n3)
{
int i,j,k;

for ( k=1 ; k <= n3; k++ )
for ( i=1 ; i <= n1 ; i++ )
{
c[i][k] = 0.0;
for ( j=1 ; j <= n2 ; j++ )
c[i][k] = c[i][k]+a[i][j]*b[j][k];
}
}

pascal void mat_mul_c_opt
(extended c[50][], long nc,
 extended a[50][], long na,
 extended b[50][], long nb,
 long n1, long n2, long n3)
{
register int i,j,k;
register extended sum;

for ( k=1 ; k <= n3; k++ )
for ( i=1 ; i <= n1 ; i++ )
{
sum = 0.0;
for ( j=1 ; j <= n2 ; j++ )
sum = sum+a[i][j]*b[j][k];
c[i][k] = sum;
}
}

```

Listing 5 : inner loops compared, Nosy-disassembled

```

pascal, optimized
len_3 MOVEA.L param2(A6),A0
MOVE D6,D0
MULS #$258,D0
MOVE D5,D1
MULS #12,D1
ADD D1,D0
MOVEA.L param3(A6),A1
MOVE D5,D1
MULS #$258,D1
MOVE D7,D2
MULS #12,D2
ADD D2,D1
LEA -$264(A0),A0

```

```

FMOVE.X 0(A0,D0.W),FP0
LEA -$264(A1),A0
FMUL.X 0(A0,D1.W),FP0
FADD FP7,FP0 ; could use
FMOVE FP0,FP7 ; FADD FP0,FP7 here
ADDQ #1,D5
BVS.S 1an_5
1an_4 CMP.W van_1(A6),D5
BLE 1an_3

```

c, optimized

```

1ar_1 MOVE.L D7,D0
MOVE.L D0,D1
MULU #12,D0
SWAP D1
MULU #12,D1
SWAP D1
CLR D1
ADD.L D1,D0
ADD.L D6,D0
MOVE.L D5,D1
MOVE.L D1,D2
MULU #12,D1
SWAP D2
MULU #12,D2
SWAP D2
CLR D2
ADD.L D2,D1
ADD.L D7,D1
FMOVE.X 0(A3,D0.L),FP0
FMUL.X 0(A4,D1.L),FP0
FMOVE FP7,FP1
FADD FP0,FP1
FMOVE FP1,FP7
ADDQ.L #1,D7
1ar_2 CMP.L D7,D4
BGE 1ar_1

```

pascal, plain

```

1am_3 MOVEA.L param2(A6),A0
MOVE D6,D0
MULS #258,D0
MOVE D5,D1
MULS #12,D1
ADD D1,D0
MOVEA.L param3(A6),A1
MOVE D5,D1
MULS #258,D1
MOVE D7,D2
MULS #12,D2
ADD D2,D1
LEA -$264(A0),A0
FMOVE.X 0(A0,D0.W),FP0
LEA -$264(A1),A0
FMUL.X 0(A0,D1.W),FP0
MOVE D6,D0
MULS #258,D0
MOVE D7,D1
MULS #12,D1
ADD D1,D0
LEA -$264(A4),A0
FADD.X 0(A0,D0.W),FP0
MOVE D6,D0
MULS #258,D0
MOVE D7,D1
MULS #12,D1
ADD D1,D0
LEA -$264(A4),A0
FMOVE.X FP0,0(A0,D0.W)
ADDQ #1,D5
BVS.S 1am_5
1am_4 CMP.W vam_1(A6),D5
BLE 1am_3

```

c, plain

```

1ao_3 MOVE.L D5,D0
MOVE.L D0,D1
MULU #12,D0
SWAP D1
MULU #12,D1
SWAP D1
CLR D1
ADD.L D1,D0
ADD.L D6,D0
MOVE.L D7,D1
MOVE.L D1,D2
MULU #12,D1
SWAP D2
MULU #12,D2
SWAP D2
CLR D2
ADD.L D2,D1
ADD.L D6,D1
MOVEA.L param3(A6),A0
MOVE.L D5,D2
MOVE.L D2,D3
MULU #12,D2
SWAP D3
MULU #12,D3
SWAP D3
CLR D3
ADD.L D3,D2
ADD.L D7,D2
FMOVE.X 0(A4,D1.L),FP0
FMUL.X 0(A0,D2.L),FP0
FADD.X 0(A3,D0.L),FP0
MOVE.L D5,D0
MOVE.L D0,D1
MULU #12,D0
SWAP D1
MULU #12,D1
SWAP D1
CLR D1
ADD.L D1,D0
ADD.L D6,D0
FMOVE.X FP0,0(A3,D0.L)
ADDQ.L #1,D7
1ao_4 CMP.L D7,D4
BGE 1ao_3

```

Fortran, optimized

```

1ah_3 MOVE -172(A6),D2
EXT.L D2
EXT.L D2
SUB.L -142(A6),D2
MULS.L #12,D2
MOVE.L D2,D0
MOVE -170(A6),D2
EXT.L D2
EXT.L D2
SUB.L -130(A6),D2
MULS.L -134(A6),D2
ADD.L D0,D2
MOVEA.L 32(A6),A0
ADDA.L D2,A0
FMOVE.X (A0),FP7
MOVE -170(A6),D2
EXT.L D2
EXT.L D2
SUB.L -118(A6),D2
MULS.L #12,D2
MOVE.L D2,D1
MOVE -168(A6),D2
EXT.L D2
EXT.L D2
SUB.L -106(A6),D2
MULS.L -110(A6),D2
ADD.L D1,D2

```

```

MOVEA.L 24(A6),A1
ADDA.L D2,A1
FMUL.X (A1),FP7
FADD.X -94(A6),FP7
FMOVE.X FP7,-94(A6)
ADDQ #1,-170(A6)
SUBQ.L #1,D5
BGT 1ah_3

```

Fortran, plain

```

1ae_3 MOVE -164(A6),D2
EXT.L D2
EXT.L D2
SUB.L -134(A6),D2
MULS.L #12,D2
MOVE.L D2,D0
MOVE -162(A6),D2
EXT.L D2
EXT.L D2
SUB.L -122(A6),D2
MULS.L -126(A6),D2
ADD.L D0,D2
MOVEA.L 32(A6),A0
ADDA.L D2,A0
FMOVE.X (A0),FP7 ; get a(i,k)
MOVE -162(A6),D2
EXT.L D2
EXT.L D2
SUB.L -110(A6),D2
MULS.L #12,D2
MOVE.L D2,D1
MOVE -160(A6),D2
EXT.L D2
EXT.L D2
SUB.L -98(A6),D2
MULS.L -102(A6),D2
ADD.L D1,D2
MOVEA.L 24(A6),A1
ADDA.L D2,A1
FMUL.X (A1),FP7 ; multiply by b(i,k)
MOVE -164(A6),D2
EXT.L D2
EXT.L D2
SUB.L -158(A6),D2
MULS.L #12,D2
MOVE.L D2,D1
MOVE -160(A6),D2
EXT.L D2
EXT.L D2
SUB.L -146(A6),D2
MULS.L -150(A6),D2
ADD.L D1,D2
MOVEA.L 40(A6),A1
ADDA.L D2,A1
FADD.X (A1),FP7 ; add c(i,k)
MOVE -164(A6),D2; this
EXT.L D2; whole
EXT.L D2; stuff
SUB.L -158(A6),D2; is
MULS.L #12,D2 ;
MOVE.L D2,D1 ; R
MOVE -160(A6),D2; E
EXT.L D2; D
EXT.L D2; U
SUB.L -146(A6),D2; N
MULS.L -150(A6),D2; D
ADD.L D1,D2 ; A
MOVEA.L 40(A6),A1 ; N
ADDA.L D2,A1 ; T !!!
FMOVE.X FP7,(A1) ; put back c(i,k)
ADDQ #1,-162(A6)
SUBQ.L #1,D5
BGT 1ae_3

```



Programmer's Workshop

SADE™ Debugging



Joe Pillera
Ann Arbor, MI
Volume 5, Number 9

Debugging in MPW With SADE™

Joe Pillera is a Scientific Staff Member at Bell Northern Research (BNR) in Ann Arbor, Michigan. Prior to this he was a Systems Analyst at the Ford Motor Company. He earned his B.S. in Computer Science from Eastern Michigan University, and is now a Master's of Computer Science degree candidate at Wayne State University. His primary area of research is object oriented programming and software engineering.

Introduction

A good debugger is a programmer's best friend. One of the serious drawbacks to MPW (Macintosh Programmer's Workshop) software development has always been a lack of a source level debugger (no, I don't consider MacsBug hexadecimal dumps debugging!). Until now. Introducing SADE™ - the Symbolic Application Debugging Environment. This is Apple's new source level debugger for MPW 3.0 and beyond.

In this article I plan to cover the following:

- An overview of SADE.
- SADE in action - debugging a sample C program.
- A numerical integrator - pushing the SADE script language to the limit.
- Talking straight - the good, the bad and the ugly.

Note that even though this is not a 'comparison' article, I will occasionally mention THINK's Lightspeed™ debuggers (because of their excellent performance), as a frame of reference in discussing SADE.

What Is SADE, Anyway?

SADE is not an MPW tool, nor is it physically part of the MPW shell. It is simply a stand-alone application that communicates with your target program under Multifinder. As in THINK's Lightspeed C, this debugger must be operated under Multifinder, to allow interprocess communication. Two or more megs of memory are strongly recommended; however, the manual does indicate you can squeeze by with one meg. The only special requirements to run SADE are:

1. You must be using Multifinder 6.1b7 or later.
2. The (Pascal or C) compiler must be instructed to generate 'symbol' files during program compiles.

One of the more interesting design decisions Apple made was to make SADE appear almost identical to the MPW Shell. This has both an advantage and a disadvantage: the advantage is that of instant familiarity to existing MPW programmers; the disadvantage is that this precludes the use of a user-friendly, animated interface (such as the debuggers used in THINK's Lightspeed C and Lightspeed Pascal).

Please refer to Figure 1 for a closer look at SADE's

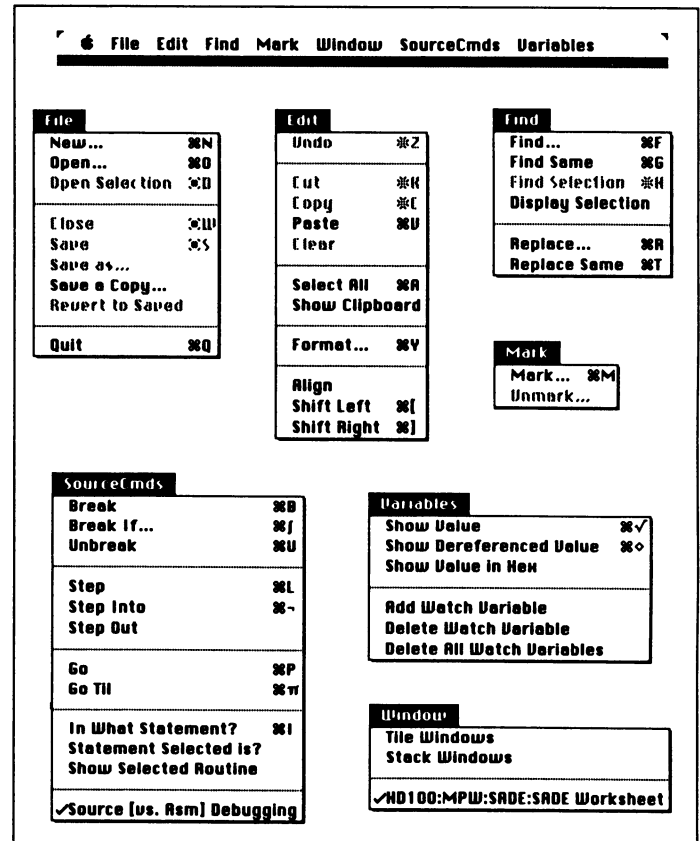


Figure 1: The SADE Menu System

functionality.

Wade Through Your Code With SADE

The most fundamental operation in using a debugger is setting a breakpoint in the source code. Because SADE is so sophisticated, there are *four* ways to do this:

1. Enter the break command from the SADE command interpreter. The syntax (for debugging a C program) is:

```
break <function name>.<(<line number>)>
```

2. Call the function SysError from within the C code itself. The syntax of this approach is:

```
#include <Errors.h>
...
SysError(<integer from 129..32511>).
```

3. Press the SADEKey (Command-Option-<period>).
4. Open a source code file (as read-only from SADE), and

use the mouse to select a line of code. Then execute 'Break' from the SourceCmds menu.

Method number 1 is fine if your functions are small, otherwise counting the individual lines of code could prove cumbersome.

Why does method number 2 work? When using version 6.1b7 and above of Multifinder, MC68000 exceptions are passed to SADE, if present; if SADE isn't a concurrently executing task, then a system exception occurs.

Personally, I think method number 3 is somewhat satirical; trying to guess which machine instruction you are now (ahem, *were*) executing can cause brain damage.

Method number 4, of course, appears to be the most user-friendly of them all.

So in practice, it would seem that all methods except number three can be easily used to set break points in your own code. Refer to Figure 2 for an example of starting up SADE and setting breakpoints. For an illustration of SADE stopping at a breakpoint, please refer to Figure 3. The MPW C source listings will also provide more information on how to set breakpoints within your own code.

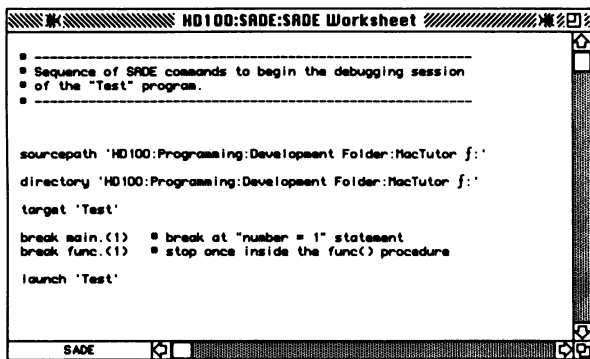


Figure 2: Launching the 'Test' program from SADE

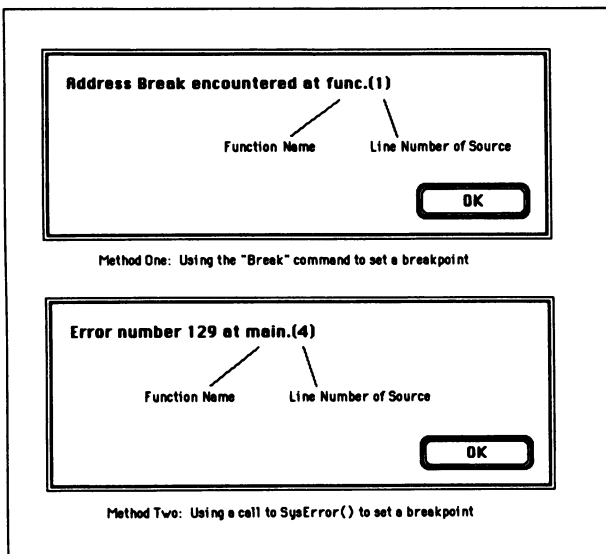


Figure 3: Encountering source code breakpoints with SADE

One interesting thing I noticed is that a SysError() breakpoint will stop *after* that line of code is executed, whereas setting a normal breakpoint will stop *before* that line of code.

SADE has complete facilities for keeping track of variable values. You can either perform a *Show Value*, which shows the contents of the variable of your choice, or you can use the *Watch* feature. This feature is analogous to THINK's Lightspeed Pascal's 'Observe' window: every time a variable's value changes, it is updated in a display window for you. Please refer to Figure 4 for an example from the 'Test' program.

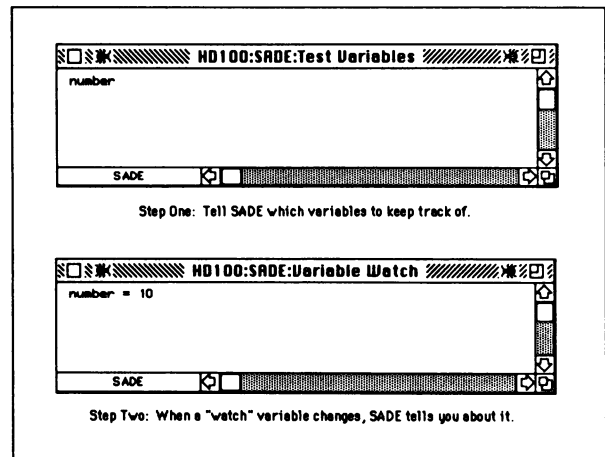


Figure 4: Observing variables change state

And of course, you have complete code stepping facilities: single step lines of code, and even control whether or not you step into procedure calls.

Program Your Bugs Away

All of the aforementioned features can be found in most other debuggers. However, the feature that sets SADE apart from any other debugger I've used is its internal programming facility (henceforth, I'll refer to SADE programs as 'scripts', to differentiate these from your C & Pascal programs). You can actually write SADE procedures and functions to automate debugging tasks. And if that's not enough, how about a slew of predefined procedures (even *printf*) and reserved system variables? Furthermore, complete expression evaluation is possible, including pointer manipulation, bit-wise operations, and string manipulation.

The script facility of SADE is so extensive, it strongly resembles Pascal. Not only that, SADE procedures and functions are fully capable of recursion. Unlike Pascal, however, all variables are dynamically typed - based on their use in the program; this would appear to be the result of an interpretative implementation of this language.

To execute a SADE script, enter the script name (followed by any arguments) and press 'enter' from the numeric keypad. Thus, script invocation is identical to executing MPW tools. Before a SADE subroutine can be run, the file that contains it must first be loaded into memory; this is done as follows:

```
execute '(<Full HFS Path>:<File Name>'
```

As an example, I've developed a general purpose numerical integration script, which when given any arbitrary function $f(x)$, will find the (approximate) integral bounded between two values 'a' and 'b'. The foundation to my algorithm is the SADE *eval* function; this function takes a string representing an expression, and then proceeds to parse it in search of an answer. For example:

```
define x = 5
eval('x * 2')
```

When executed at the SADE command line, this would return '10' as the result.

The technique of integration I use is called Simpson's Rule, which partitions an interval from 'a' to 'b' an even number of times (i.e. $N=2M$ partitions). We then approximate the graph of $f(x)$ by M segments of parabolas; the technique is shown below:

$$\int_a^b f(x) dx = \frac{1}{3} \left[f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) + \dots + f(x_{2M}) \right] \Delta x$$

In my case, I partition each region between n and $n+1$ four times, giving Δx the value of 0.25; integration then simply becomes a matter of alternating the coefficients of the intermediate data values.

Unfortunately, it wasn't that easy - because I wanted to be able to integrate any continuous function I could think of, not just the simple ones. As it turns out, simple mathematical functions (such as x^{**n}) are not known to SADE, so I had to write my own.

How did I do this for polynomials? Using something called a Taylor Expansion, it is possible to calculate x^{**n} (x to the n th power), where either 'x' or 'n' can be floating point values. Taylor Expansions are open-ended functions, so it is necessary to determine a stopping criteria that is both accurate and efficient. The Taylor Expansion for x^{**n} is shown below:

$$x^n = 1 + n \ln(x) + \frac{(n \ln(x))^2}{2!} + \frac{(n \ln(x))^3}{3!} \dots$$

For example: because both 'x' and 'n' can be floating point numbers, integrating the square root of 'x' is simply a matter of integrating $x^{**}(1/2)$ power.

Just one more problem - $\ln(x)$ is not known to SADE either, so here's the approximation for it:

$$\ln(x) = 2 \left[\frac{x-1}{x} + \frac{1}{3} \left(\frac{x-1}{x} \right)^3 + \frac{1}{5} \left(\frac{x-1}{x} \right)^5 \dots \right]$$

This approximation, of course, is only valid for $x > 0$. For an example of a simple polynomial integration, please refer to Figure 5.

However, before you throw away your favorite numerical analysis package, I should warn you that I found SADE to have problems maintaining floating point precision. Take a look at my

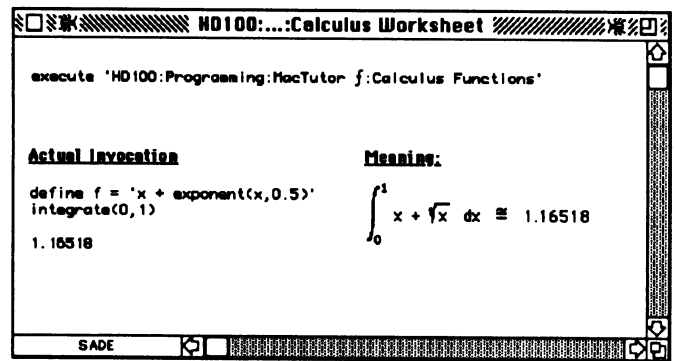


Figure 5: Numerical Integration Using My SADE Script

exponent script: I noticed that using tricks such as intermediate variables for subexpressions made the difference between a correct answer and something way out in left field. SADE's natural habitat seems to be integers, which would make a lot of sense seeing how it's a debugging environment. Therefore, even though Simpson's Rule will (theoretically) work on any continuous function, my SADE script will only work in cases when SADE handles floating point calculations correctly. We should remember, that SADE is a debugging environment, so I don't consider this to be a drawback.

If all these features still aren't enough, SADE even provides high level calls to prompt the user with dialog boxes, add custom menus, and even generate music! In short, SADE has every power-tool a programmer could ever ask for in a debugger.

Straight Talk About SADE

SADE is by far the most sophisticated and user-extendable debugger I've ever used - but I do have one complaint: its user interface. This drawback is caused by SADE's own complexity: at times it seems to offer you *too many* features. All of these features in turn dictate a user-interface that's not as elegant as the Lightspeed debuggers. If SADE were made to be more user-friendly, even at the expense of some of its bells and whistles, Apple would have a world-class debugger in their hands.

The End...

MacsBug, MPW, Multifinder and SADE are trademarks of Apple Computer, Inc. Lightspeed is a registered trademark of Lightspeed, Inc.

The opinions expressed here are solely those of the author, and not of Bell Northern Research Inc., or its employees.

BIBLIOGRAPHY

Apple Computer, Inc. (1989) *SADE: Symbolic Application Debugging Environment*, Apple Programmer's and Developer's Association, Cupertino, California.

Beyer, William H., PhD. (1978) "Series Expansion," p.387 in *Standard Mathematical Tables - 25th Edition*, CRC Press Inc., Boca Raton, Florida.

Shenk, Al, PhD. (1979) "Techniques of Integration," pp. 424-425 in *Calculus and Analytic Geometry - 2nd Edition*, Scott, Foresman and Company, Glenview, Illinois.

Symantec, Inc. (1989) "Debugging Programs," pp. 103-110 in *THINK's Lightspeed Pascal User's Manual*, The Symantec Corporation, Cupertino, California.

Symantec, Inc. (1989) "The Debugger," pp. 129-145 in *THINK's Lightspeed C User's Manual*, The Symantec Corporation, Cupertino, California.

===== The Integrator Script =====

```
# -----
# A simple but powerful use of the SADE™
# script language to perform numerical
# integration on almost any function.
# -----
# Copyright © Joe Pillera, 1989
# All Rights reserved
# -----

# -----
#          func integrate(a,b)
# -----
# Integrate a global function 'f(x)' from
# a to b.
# -----
# INPUTS:
# a -> integer or floating point
# b -> integer or floating point
# -----
func integrate(lower,upper)
    define factor, result, index, x
    index := lower
    factor := 2
    result := 0
    while index <= upper
        x := index
        if (index = lower) or (index = upper) then
            result := result + eval(f)
        else
            result := result + (factor * eval(f))
        end
        index := index + 0.25
        if factor = 2 then
            factor := 4
        else
            factor := 2
        end
    end
    end
    return (1.0/12.0) * result
end

# -----
# func ln(x) -> Compute natural log of x
# -----
# INPUTS:
# x -> integer or floating point
# -----
func ln(x)
    define result, term, index, frac
    term := (x - 1.0) / (x + 1.0)
    result := term + (0.3333 * power(term,3))
    index := 5
    while index <= 25 do
        frac := 1.0 / index
        result := result +
            (frac * power(term,index))
        index := index + 2
    end
end
```

```
end
return (2.0 * result)
end

# -----
# func exponent(a,x) -> Compute a ** x
# -----
# INPUTS:
# a -> integer or floating point
# x -> integer or floating point
# -----
# NOTE: Use power(a,x) if 'x' is a integer,
#        because that routine is faster.
# -----
func exponent(a,x)
    define result, iteration, term
    define numerator, denominator
    term := x * ln(a)
    result := 1.0 + term
    for iteration := 2 to 15 do
        numerator := power(term,iteration)
        denominator := fact(iteration)
        result := result +
            (numerator / denominator)
    end
    return result
end

# -----
# func power(x,n) -> Compute x ** n
# -----
# INPUTS:
# x -> integer or floating point
# n -> integer
# -----
# NOTE: Use iteration for speed.
# -----
func power(x,n)
    define index, result
    if n = 0 then
        return 1
    elseif n = 1 then
        return x
    else
        result := x
        for index := 2 to n do
            result := result * x
        end
        return result
    end
end

# -----
# func fact(n) -> Compute n!
# -----
# INPUTS:
# n -> integer
# -----
func fact(n)
    if n = 0 then
        return 1
    elseif n = 1 then
        return 1
    else
        return (n * fact(n-1))
    end
end

===== Main.C =====

#include <Errors.h>

int number;

main()
```

```

{
/* Set breakpoint #1 here (from SADE) */
number = 1;

/* Now pass control to func, and let */
/* it alter the "number" variable. */
func();

/* Hard code breakpoint #3 here */
SysError(129);
}

===== Func.C =====

extern int number;

void func()
{
/* Set breakpoint #2 here (from SADE) */
number = 10;
}

===== Test.R =====

#include "SysTypes.r"

resource 'vers' (1) (
    0x1,
    0x00,
    release,
    0x0,
    verUS,
    "",
    "1.0 / MacTutor Magazine"

```

```

);

resource 'vers' (2) (
    0x1,
    0x00,
    release,
    0x0,
    verUS,
    "",
    "For Use With MPW SADE™ Only!"
);

===== Make File =====

Test ff Test.make test.r
    Rez test.r -append -o Test
func.c.o f Test.make func.c
    C -sym on func.c
main.c.o f Test.make main.c
    C -sym on main.c

SOURCES = test.r func.c main.c
OBJECTS = func.c.o main.c.o

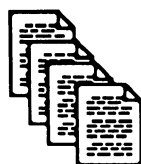
Test ff Test.make {OBJECTS}
    Link -w -t APPL -c '????' -sym on -mf d
    {OBJECTS} d
    "(Libraries)"CRuntime.o d
    "(Libraries)"Interface.o d
    "(Libraries)"StdCLib.o d
    "(Libraries)"CSANELib.o d
    "(Libraries)"Math.o d
    "(Libraries)"CInterface.o d
    -o Test

```



Developer's Notes

New Typographic Frontier



Frank Alviani
Waukegan, IL
Volume 5, Number 10

Apple's New Typographic Universe

At the 1989 Developers' Conference, Apple revealed an entirely new typographic universe to 1500 eager supporters. The combination of a new font technology, a greatly enhanced line layout manager, and an entirely new printer driver architecture promises to make the Macintosh the premier machine for print-oriented graphics, and open new opportunities for Macintosh developers. The three features are closely related and need to be discussed together to understand the full impact.

Why Outline Fonts?

The two great advantages of outline fonts are the ability to display them nicely at any point size, and their compactness when compared to a reasonably large set of bitmaps. For example, the full set of bitmaps for the (exceptionally nice) Clairvaux font occupies over 200K, while the outline description is estimated to average 40-60K. They are not magic, however - just a shorthand method for describing bitmaps.

Ultimately, all raster devices actually use bitmaps to place the characters of a font into an image; it doesn't matter whether you're talking about a 2540 dpi Linotronic imagesetter or a 72 dpi 128K Macintosh. What outline fonts provide is an efficient way to generate a bitmap of a specified character at an arbitrary size that can then be saved and used whenever needed. The "quality" of an outline font system depends on two factors:

1) *The quality of the generated bitmap.* If the bitmap generated by the process is not visually appealing, nothing else matters.

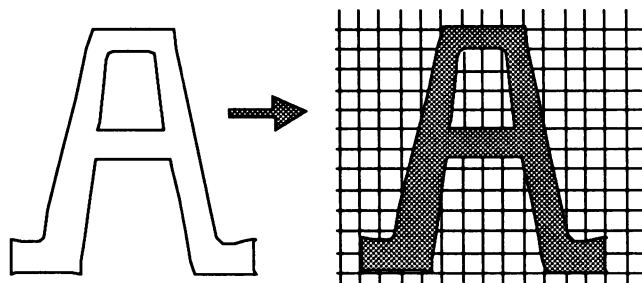
2) *The speed of the generation process.* If it takes forever to create a bitmap of a character, who would use that process?

The process of generating a bitmap consists, for both the Apple and Postscript systems, of two basic parts: creating the basic bitmap, and making any small adjustments necessary for the best appearance at small point sizes. Creating the bitmap from an outline description can be visualized as drawing the outline over a grid of small squares, and filling in all the squares that fall at least partially within the outline.

If you carried out this little exercise, you were probably bothered by some of the effects of filling in squares that only partially fell within the outline. This demonstrates the problems encountered with small type on machines with relatively low resolution (large pixels). Small point sizes are a problem for all systems simply because:

1) With low resolution and small point sizes, a single pixel occupies a significant percentage of an image. Therefore, it is important to make sure every pixel is optimally placed.

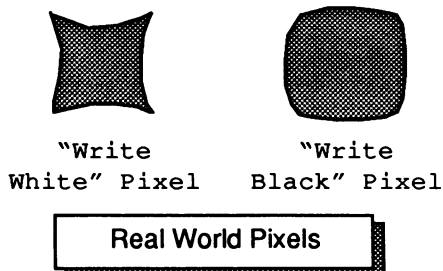
2) "The jaggies" are unavoidable, but a visual decision on where a pixel looks best sometimes differs from a purely mathe-



Character Outline Projected onto a Device Specific Grid to create a Bitmap

matical decision. Naturally, this conflict is more significant when there are only a few pixels to play with.

3) While the idealized pixel is generally square, real world pixels never are, whether you are dealing with a CRT or a LaserWriter imaging drum. This means that corrections must be made to account for the visual effects of the actual pixel shapes on the equipment being used. There are two basic shapes of "real" pixels:



The reason for the different shapes is that in reality all pixels tend to be convex; however, when convex white pixels are drawn over black to define a black pixel, the black pixel is left with concave edges.

One of the major advantages of the Apple outline font system from the artist's viewpoint is that the entire "hinting" language needed to make the bitmap adjustments is precisely defined in complete detail; this is in contrast to the undocumented system used by Adobe, which has historically given them a small but important advantage over their competitors in image quality. The features included in the hinting language were chosen to include all the features used by the major type foundries, thus making the conversion of fonts by those type foundries to the Apple format as accurate as possible. The result should be a steady flow of very high quality fonts once System 7.0 is in place. In fact, Apple has been working with most of the major type

foundries in developing their system.

Another result of Apple's decision to publish the complete specifications of its outline font system is that we can expect to see one or more font editors enter the market that will allow the artist to completely specify the hinting required for the best possible quality at small point sizes. This levels the playing field and places the competitive emphasis on artistic ability rather than secret technology, which will be to everyone's benefit in the long run.

Apple's font descriptions, using quadratic splines, are designed to be simple enough that it is reasonable to use them for interactive displays such as the Macintosh itself; the original Postscript model (which uses Bezier curves - a more complex equation form) is oriented towards printing engines. The fundamental approach is to generate the required bitmaps for an entire size when first requested and cache them; thus, the overhead is only incurred once. Once the bitmaps have been built, the Macintosh text operations are exactly as fast as when using the "old" fonts. **"Old style" fonts are completely compatible with the new system.** This means that you won't need to throw out all those nifty display fonts you already have when you install System 7.0

What's so Important about the new Line Layout Manager?

While the new font manager provides very precise control of the shape of letters, the new line layout manager provides some very powerful and impressive capabilities in conjunction with the new font descriptions. One of the primary goals of this manager is to work together with the Text Edit package, using information provided in the font descriptions, to greatly enhance the line layout facilities available to the developer and user. The preliminary description handed out at the Developers' Conference noted that many of the features of the Layout Manager, while optional in English and other roman-script languages, are mandatory for acceptable text display in many other languages.

There are two basic types of features implemented in the Layout Manager: *Positional features*, and *Non-positional features*. As you probably have deduced from the names, the positional features affect horizontal and vertical positioning of characters, while non-positional features primarily affect character shaping. The distinction will become clearer when I list some of the individual items in each group.

Positional Features. These include the types of manipulations traditionally considered part of "fine typography", such as

- **Kerning:** This is the most basic positional manipulation. It involves moving a pair of characters (such as T-y) closer together to improve the appearance of the text. When "negative kerning" is done it is often termed letterspacing.

- **Optical alignment:** This is the adjustment of positioning so that margins are optically correct. The problem arises because of the space surrounding a character that is part of its definition.

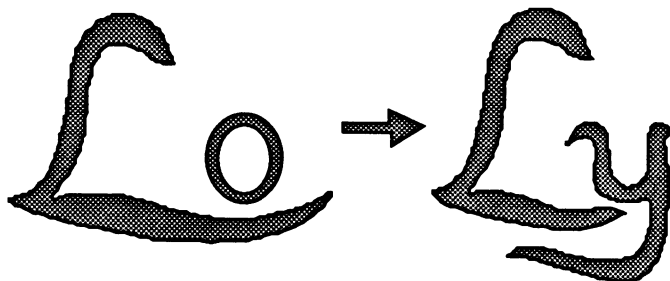
- **Hanging punctuation:** Punctuation marks are often placed "outside" the margins since they are optically small so that the optical flushness of the margins looks better. A line ending with a period, for example, seems to be visually slightly ragged.

- **Justification and centering:** Just what they sound like.

Non-positional features. These are features that concentrate on refining the shape of letters, primarily in response to the immediate context for each letter. Many of these features were difficult to implement, to say the least, with a traditional typesetting system. A partial list of these features includes:

- **Ligaturing:** This is the combination of two letters into a single graphic, and was traditionally done both for more compact layout and for a more elegant appearance. Common ligatures include "æ", "œ", etc.

- **Contextual forms:** As the name suggests, this is a technique for shaping a letter to look best in a very specific context. For example, a capital 'L' with a long low horizontal stroke that goes under the succeeding character is very attractive as long as the succeeding character doesn't have a descender. With contextual reforming, the Layout Manager can choose a non-underlining form if the succeeding character has a descender. For example:



Contextual Reforming of a letter

- **Applied marks:** These are typically various forms of accents, such as the tilde (~) or accent ('). There are various styles of positioning these.

- **Reordering of graphics:** In certain non-roman languages, such as Sanskrit, the order of characters is dominated by their shape, so that the visible order of certain character combinations may not correspond to their logical order. This reordering, fortunately, follows firm rules and not artistic whim.

It should be emphasized that the modifications that can be applied by the Layout Manager happen fast enough to be very real-time. Many of the features of the Layout Manager are specified in the font description, so that the type artist can have precise control over contextual reformatting, for example, which is automatically applied without any intervention by the application.

Certain Text Edit functions are now better carried out by the Layout Manager. In particular, these include the text measuring and caret management functions, which are very dependent on the higher precision supported. Highlighting can now be non-rectangular and non-contiguous, for example. As in FullWrite, a caret in italic text is slanted at the proper angle, which actually serves to make placement much easier.

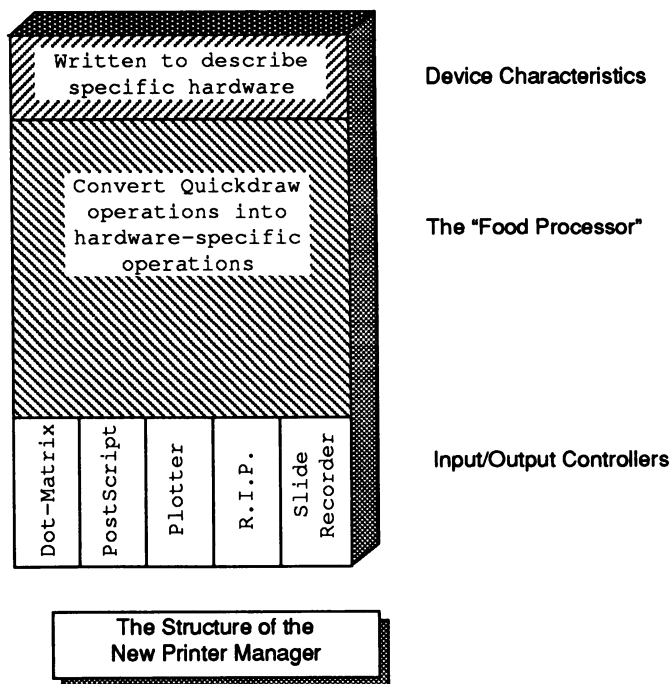
The combination of the new font descriptions and the power

of the Layout Manager gives the Mac user the power to create text with every typographic refinement available to the traditional typesetter, and then some. Up until now, however, everything I've talked about seems to have been limited to the screen - so what good does it do when the ink hits the paper?

The New Printing Architecture - Getting It All Down In Black and White.

And other colors as well, of course... Printer drivers have always been one of the sore spots in the Macintosh architecture, being extremely difficult to write well and tending towards instability. Apple's stated goals for System 7.0 required that the entire internal design approach for these drivers be changed. As a result, the new drivers are totally new code, with considerable advantages.

The new driver structure is explicitly layered into: (1) An I/O layer specialized for a number of device types, (2) A "Food Processor" layer that handles the bulk of the work of dicing, slicing, and chopping the user's requests into device-specific byte streams, and (3) A specification layer, making up about 10% of a driver, that is actually written by the driver author to describe the hardware-dependent aspects of the device and thus convert generic requests by the user into specific calls to the food processor, where the actual conversions take place.



As shown in the diagram, there are several different devices supported. According to an experienced printer-driver author I spoke to, driving a film recorder is actually the most difficult, due to extremely stringent timing constraints. Perhaps the most interesting from a marketing viewpoint is the "R.I.P." - the Raster Image Processor that forms the heart of the modern typesetter, or

imagesetter. While it was firmly stated that Apple would never license the technology at the core of the image creation process, a Mac II-class machine with an EtherNet card would be very viable as the heart of a relatively low-price high resolution typesetter that could very effectively compete with a Postscript typesetter. I don't expect that it will take too long for such a device to appear on the market...

Unlike the previous structure, the new printer manager is explicitly designed to simplify the task of writing output drivers as much as possible. The authors claimed at the Developers' Conference that the average time to create a printer driver would decrease from months to days, and that they had created several test drivers in less than a day.

The motivation for the support of 3rd party devices is simple: Apple has decided that support for the widest possible variety of printers, plotters, etc. is necessary to make the Mac the printing platform of choice. As conclusive evidence of this, they revealed that the "print shop" is now committed to actively supporting everyone who wishes to make a Mac-compatible printer - even though they believe that the LaserWriter will lose market share as a result.

There are a number of additional major enhancements to the printing capabilities of the Mac. Among the more important ones are:

- The ability to redirect spooled output after before printing. This means that if a particular LaserWriter turns out to be extremely busy, you can send a waiting document to another printer without having to reprint it. For this alone, most people would be ecstatic.

- The ability to redirect a printed document to a printer of a different type without having to reprint it, and without having the layout recalculated. This may require some cooperation on the part of the application authors; the basic idea is the the application will output both a "final" quality and a "draft" quality version at the same time; the draft quality component will use the layout information for the final quality portion. This means that you can choose a LaserWriter for the final output device, but send the document to an ImageWriter II and get an accurate idea of the final appearance.

- The ability to keep a spooled document after it's been printed, so that munged pages could be reprinted if necessary.

- Further off in the future (post-System 7.0), support for high-volume remote printing stations that would include multiple output queues (so you could just say "Use the 1st available LaserWriter" rather than having to explicitly pick one), good status reporting, and other features designed to work in an environment such as many large offices.

- Greatly redesigned user-interfaces for the printer dialogs, allowing you to choose a printer from a printer dialog, set a custom page size for a job, and support multiple page types in a single job (such as a #10 envelope and legal paper), among other things.

From the viewpoint of an application developer, there were several critical matters that are being handled properly in the new

approach. First, the application's interface to the printer drivers is structurally identical to the way matters work today, although routine names have been changed to protect the innocent. This means - hopefully - that a simple conversion will be all that's needed to become printing compatible with System 7.0. Secondly, recognizing that the new functionality is too radically different from that in pre-System 7.0 systems, Apple is making no attempt to make the new drivers work in older systems - which means that past mistakes are not going to be anchors on the new drivers. The downside is that no existing drivers will work under System 7.0, but changing printer drivers is a fairly trivial matter. Finally, the print shop claimed the new drivers are noticeably faster than the older drivers - which cheered everybody.

Perhaps the most important fact was that the print samples shown were of uniformly excellent quality. I once worked for a typesetter manufacturer, and the output I saw at the Developers' Conference was on a par with "real" typesetters.

What's ahead in the future?

As if all these developments weren't enough, a little information about future developments leaked out - just enough to tantalize. The Remote Printing Stations were fairly explicitly described, but we were cautioned that they wouldn't be ready in time for System 7.0. It was stated that the new font system will support rotated text, but that the *current* implementation of Quickdraw couldn't handle it. And finally, somebody in development was overhead mentioning that the resolution-independent version of Quickdraw would support the rotated text, but that an

implementation compatible with existing applications was proving difficult and was "a ways off".

So much for my Robert Cringely imitation; now back to reality...

What do these developments actually mean?

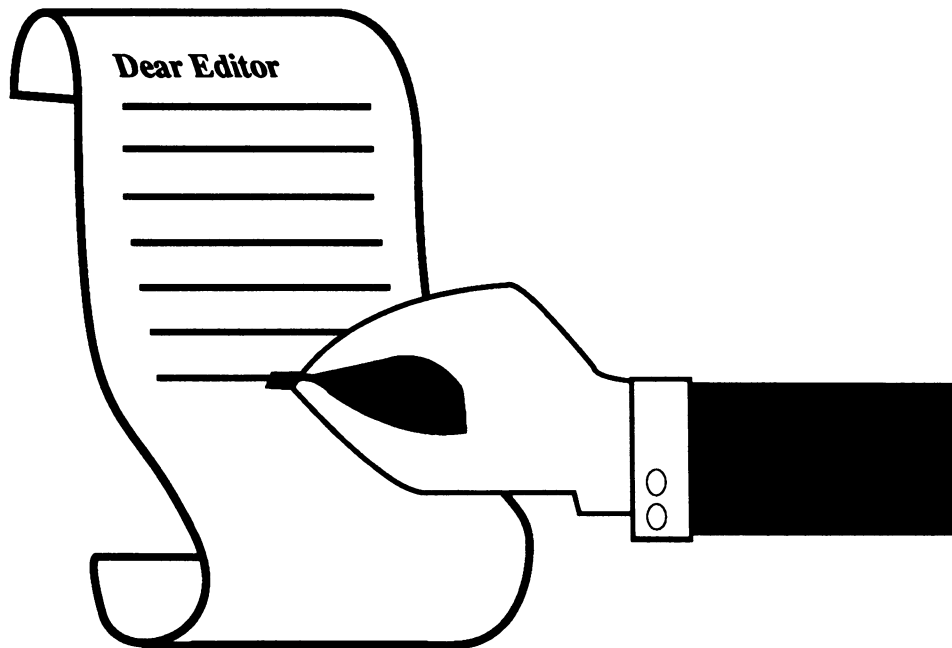
The most important outcome from these new developments is that finally a single imaging model will be usable on everything from the Mac screen to a high-resolution imagesetter, eliminating the mis-matches that now make WYSIWYG really mean "What You See Is Almost What You Get". This gives Apple a dramatic advantage over competing machines, where there are always differences between the screen and hardcopy graphic models, with the ensuing annoyances.

From the users' viewpoint, the most noticeable immediate change will be that the quality of text on the screen is going to improve dramatically. This should make the Mac even more competitive against the MS-DOS machines. Once the user starts printing documents, the new speed and flexibility should be very pleasing.

It should be emphasized that your Postscript fonts and printers don't become instantly obsolete with System 7.0. The Postscript drivers are intelligent enough to determine if you are using a resident font and to use that directly, rather than try to stuff the Apple version of Courier into a PostScript printer.

Ultimately, we all win - better output and applications for the user, and a larger market for the developer!





Letters Column

Letters



David E. Smith
Editor & Publisher
MacTutor

Volume 5, Number 1

Overriding ROM Resources Ronny Zulalkha Sydney, Australia

When Apple upgraded to the 128K ROM, some of the extra space was filled with ROM resources that were previously only held in the system file. This saves time in that these resources aren't loaded from the system file when needed, and saves space in that RAM isn't taken up by them.

If one is designing a resource that is to replace one in the ROM, merely placing the new resource in the system file is not sufficient. As described in IM IV-20, one needs an 'ROvr' resource ID 0, which my system file (4.3) already had. This resource startup scans the system file for 'ROv#' resources of any ID, which indicates what resources to override. The first word of a 'ROv#' resource is the version number of ROM to override, and is \$75 for the 128K ROM. For the SE and Mac II, it would be reasonable to assume that the version numbers listed in IM V-xvii would apply, \$76 and \$78 respectively. In fact, the routine in the 'ROvr' resource checks against the version word at location 8 higher than the start of the ROM. The high order byte is 0 for the 128K ROM, 1 for the Mac II and 2 for the SE, while the low order byte is the ROM version number. Putting this together, we get \$276 for SE and \$178 for the Mac II

Optimized String Comparisons John S. Stokes III San Diego, CA

My routine to compare two Pascal strings uses substantially less code than the "optimized" routine published in Donald Koscheka's HyperChat column in November, 1988 issue of MacTutor. In my routine, the Pascal string length fields are compared as just another character in the comparison loop instead of separately.

```
A0 -> Pascal string 1
A1 -> Pascal string 2
D0 <- 0 if false, 1 if true
D1 <- work
```

```
CompareString Moveq #0,D0 ; initialize false return
code
Move.B (A0),D1 ; length of the first string
Ext.W D1 ; dbra uses word
@10 Cmp.B (A0)+,(A1)+ ; match?
Bne.S @900 ; no
Dbra D1,@10 ; yes, continue
Moveq #1,D0 ; set return code to true
@900 Rts ; return
```

Power Supply Board David B. Lamkins

Canton, MA

Thanks to MacTutor, my Mac is alive and well at minimal expense following a sudden failure of the power supply board. Over a period of a couple weeks, under light use, my screen went from an occasional "blink" (accompanied by a snapping sound) to a thin vertical line. I recalled that there was a discussion of these symptoms in MacTutor, and found a detailed description of the problem and its cure in the June and July 1987 issues.

I'd like to shed some additional light on the subject, since I suspect that many owners of older Macs will face the same problem eventually. The Macintosh power supply board (which also provides the circuitry to operate the display) contains a wide range of parts, from integrated circuits to high-power components. It is tricky to solder the full range of these components on a wave solder machine — too little heat and the leads of the large components don't get properly soldered, too much heat and the delicate components get fried.

I found a crack around J1 pin 4, just as indicated in the earlier articles, and signs of developing cracks around the other pins of J1. I removed the old solder with a solder bulb and carefully resoldered all four pins. What surprised me was that I found that NO solder had flowed between the pins and the (I'm assuming — too late to check) plated through holes in the PC board. Thus, the contact area was limited to a very thin film of solder bridging the connector pin to the foil on the PC board. It's no wonder there was local heating leading to a joint failure!

I also found a bad solder joint on one of the leads of a coil in the same horizontal sweep circuit, and resoldered that as well. In this case, it looked like the original connection suffered from either improper cleaning of the coil lead, or too little flux and heat during soldering. A quick check of the solder joints on the remaining showed a potential problem on the end pins of the ten-pin power connector. Note that none of these problems were plainly visible to the naked eye — a magnifying glass was required

I found that the hardest part of the repair was locating the appropriate tools to open the Mac's case. The readily available MacSnap Toolkit is not worth the expense — a long piece of allen wrench stock, which fits only some of the screws, is supplied in lieu of a Torx screwdriver. I returned this immediately for a refund. Local hardware, auto supply, appliance service, and specialty tool stores (I tried at least fifteen altogether) failed to produce a long-shafted Torx T-15 screwdriver. Least helpful was the service department of the local Apple dealer — "Sorry, they're made exclusively for Apple. You can't buy one unless you're an authorized Apple repair center." Finally, I turned to the MacTutor advertisers and called Computer Quick. For \$20, these folks sold me a toolkit (separate screwdriver and case

spreader) as good as any seen in an Apple service department.

My Mac started life three and a half years ago as a 512K. A year later, it was upgraded to a 512 enhanced, and stuffed with 4 megs of RAM and a piezo fan a year after the first upgrade. I have never had even a hint of a power supply problem during all this time. The time from onset of the first "glitch" to total failure was only a couple of weeks of light use —perhaps twenty hours of power-on time. From what I've seen, I strongly recommend that owners of earlier Macs take very seriously any signs of trouble with their display.

Whatever you do, don't "fix" a jittery display by banging on the side of the case, as I've heard some people will do —this will only prolong the symptoms and probably will cause more extensive damage due to internal heating of bad solder joints. If you see the onset of this problem, you probably shouldn't leave your Mac unattended while powered on. I suspect that the failure of the yoke connection for the horizontal sweep could unduly stress related components, leading to more serious failures than can be recovered with touch-up soldering.

Finally, if you do experience the "thin vertical line" death of your Macintosh, check the solder joints as mentioned here. If you don't know how to solder, or have never worked on a television or video monitor circuit, enlist the help and advice of a friend who has. Failing that, you may want to consider a board exchange via some outfit like Computer Quick on the west coast, or Pre-owned Electronics in the Boston area. You'll save money and time compared to the service provided by many Apple dealers.

Linear Equations

C. H. Friend

Watford, Herts, England

The article by Dave Kelly in the September MacTutor takes a long way around to do the following, which is the way it would appear in most math textbooks about solving linear systems of equations:

```
REM Solve equation system using inversion by MAT routines
REM mod CHF: 2/10/88 for TrueBasic 2.0 on Mac II
OPTION BASE 1
DIM A(3,3), V(3,3), B(3,1), X(3,1)
REM MacTutor Sep'88 example:
REM coefficient matrix [A]:
DATA 1, -4, 4, -1, 6, -3, 1, 0, -1 ! equation coefficients
REM right-hand side vector (b):
DATA 7, 0, 7 ! equation data
REM System is [A]x(X) = (b)
MAT READ A
MAT READ B
PRINT "Coefficients:"
MAT PRINT A
REM inverse of coeff. matrix, [A]^-1
MAT V=INV(A)
PRINT "Inverse:"
MAT PRINT V
PRINT "right-hand sides:"
MAT PRINT B
REM Solution vector, (X) = [A]^-1 x (b)
MAT X = V * B
PRINT "Solution:"
MAT PRINT X
END
```

Results obtained are shown below:

Ok. run
Coefficients:

```
1      -4      4
-1     6      -3
1       0     -1
```

```
Inverse:
.428571      .285714      .857143
.285714      .357143      7.14286e-2
.428571      .285714      -.142857
```

```
right-hand sides:
7
0
7
```

```
Solution:
9
2.5
2
```

Ok. Bye

You also refer to Hewlett-Packard Basic, on HP9000 Series machines. The Matrix ROM on HP's smaller Series 80 machines has had an extensive set of matrix commands and functions for some time. Series 200 HP-Basic has only just acquired some of these, such as the sub-matrix copy commands. Other useful HP-Basic functions are the row and column sums. These make powerful programs very short, e.g. the following example from "Probability and Statistics for Engineers and Scientists" by Walpole and Myers (MacMillan)

```
REM Multiple Linear Regression (W&M), HP86
REM mod 25/6/88 for TrueBasic on Mac II
OPTION BASE 1
DIM D(5,15), E(15,5), F(1,5), V(15,4), X(15,5), T(5,15)
DIM AC(5,5), C(5,5), Y(15,1), Z(1,15), G(1,15), H(5,1)
DIM B(5,1), P(5,1), Y2(15)
DATA 4, 13
DATA 25.5, 31.2, 25.9, 38.4, 18.4, 26.7, 26.4,
DATA 25.9, 32, 25.2, 39.7, 35.7, 26.5
DATA 1.74, 6.32, 6.22, 10.52, 1.19, 1.22, 4.1
DATA 6.32, 4.08, 4.15, 10.15, 1.72, 1.7
DATA 5.3, 5.42, 8.41, 4.63, 11.6, 5.85, 6.62, 8.72
DATA 4.42, 7.6, 4.83, 3.12, 5.3
DATA 10.8, 9.4, 7.2, 8.5, 9.4, 9.9, 8, 9.1
DATA 8.7, 9.2, 9.4, 7.6, 8.2
REM
PRINT "Multiple Linear Regression:"
READ K,N
MAT REDIM D(K,N), E(N,K), F(1,K), V(N,K-1)
MAT REDIM X(N,K), T(K,N), AC(K,K), Y(N,1)
MAT REDIM Z(1,N), G(1,K), H(K,1), P(K,1)
MAT READ D
REM
MAT E = Trn(D)
PRINT "Data:"
MAT PRINT USING "*****.###": E
REM MAT F=CSUM (E) ! HP-BASIC
FOR J = 1 TO K ! TRUEBASIC
LET F(1,J) = 0.0
FOR I = 1 TO N
LET F(1,J) = F(1,J) + E(1,I)
NEXT I
NEXT J
PRINT "Sums:"
MAT PRINT USING "*****.###": F
MAT X=Con
REM MAT V=E(,2:K) ! HP-BASIC
FOR I = 1 TO N ! TRUE BASIC
FOR J = 1 TO K-1
```

```

      LET V(I,J) = E(I,J+1)
    NEXT J
  NEXT I
  REM MAT X(2:K)=V ! HP-BASIC
  FOR I = 1 TO N ! TRUE BASIC
    FOR J = 1 TO K-1
      LET X(I,J+1) = V(I,J)
    NEXT J
  NEXT I
  REM PRINT "Ind. variables:"
  MAT T = Trn(X)
  MAT A = T * X
  PRINT "Equations:"
  MAT PRINT USING "*****.****": A
  REM MAT Y=E(,1)
  FOR I = 1 TO N
    LET Y(I,1) = E(I,1)
  NEXT I
  MAT Z = Trn(Y)
  MAT G = Z * X
  MAT PRINT USING "*****.****": G
  MAT H = Trn(G)
  REM MAT B=SYS (A,H) ! HP-BASIC, solves linear system
  MAT C = Inv(A) ! TRUE BASIC
  MAT B = C * H
  PRINT "Coefficients:"
  MAT PRINT USING "—,*****.B "
  PRINT "residuals:"
  REM MAT Y2=Y.Y ! HP-BASIC allows dot product of vectors
  FOR I = 1 TO N
    LET Y2(I) = Y(I,1)
  NEXT I
  REM SY2=SUM (Y2) ! HP-BASIC matrix function
  LET SY2 = 0.0
  FOR I = 1 TO N
    LET SY2 = SY2 + Y2(I)
  NEXT I
  LET SY2N = G(1,1)^2/N
  LET SST = SY2 - SY2N
  MAT P = Trn(G)
  REM MAT P=B.P
  FOR J = 1 TO K
    LET P(J,1) = B(J,1)*P(J,1)
  NEXT J
  REM SBG=SUM (P)
  LET SBG = 0.0
  FOR J = 1 TO K
    LET SBG = SBG + P(J,1)
  NEXT J
  LET SSA = SBG - SY2N
  LET NDF = N - K
  LET ES2 = (SST - SSA)/NDF
  PRINT USING "SST = *****.****": SST;
  PRINT USING ", SSR = *****.****": SSR;
  PRINT USING "; s^2 = **,*****.ES2
END

```

Results are:

Ok. run

Multiple Linear Regression:

Data:

25.50	1.74	5.30	10.00
31.20	6.32	5.42	9.40
25.90	6.22	8.41	7.20
38.40	10.52	4.63	8.50
18.40	1.19	11.60	9.40
26.70	1.22	5.85	9.90
26.40	4.10	6.62	8.00
25.90	6.32	8.72	9.10
32.00	4.08	4.42	8.70
25.20	4.15	7.60	9.20
39.70	10.15	4.83	9.40
35.70	1.72	3.12	7.60
26.50	1.70	5.30	8.20

Sums:
377.50 59.43 81.82 115.40

Equations:
13.000 59.430 81.820 115.400
59.430 394.726 360.662 522.078
81.820 360.662 576.726 728.310
115.400 522.078 728.310 1035.960

377.500 1877.567 2246.661 3337.780

Coefficients:
39.157350
1.016100
-1.861649
-.343260

residuals:
SST = 438.131, SSR = 399.454; s^2 = 4.29738
Ok. Bye

Matrix statements were in the original DEC PDP Basic, developed by Kemeny and Kurtz. They were omitted from early microcomputer Basics, presumably for memory reasons. Another less useful feature of DEC Basic is this "####" print image stuff, HP-Basic uses the much neater "nD.mD" style (numbers of digits before and after the decimal point, plus other options for mantissa and exponent style, etc. *[Thank you for an excellent treatment of an interesting subject. I am very interested in covering more such areas in scientific and engineering computing. After all, computers were originally designed to solve problems, were they not? —Ed]*

Volume 5, Number 2

Update to Event Simulator Matthew Snyder Fairfield, CA

Another weekend, another thought about Event Simulator. This thought is a minor one, though, and I'm not even sure if it's significant enough to merit a change. *[This letter refers to a program published in the January 1989 issue of MacTutor. —Ed]*

The change would be a minor one, too. Right before the heading *How the Demo Works* and right after that long paragraph about memory, I would add this paragraph:

"I could have avoided the memory problem by using PPostEvent instead of Enqueue. PPostEvent takes an event type and message as arguments, and creates an event using current mouse position, modifier keys, etc., posts the event, and returns to you a pointer to the new queue element. You can then use that pointer to modify the event to your liking. However, I found this method unsatisfactory for a general library because the size of the Event Queue Buffer limits you to 20 consecutive events."

I was trying to imagine what type of critical letters I could expect from readers, and it occurred to me that anyone that's done this kind of thing before has probably used PPostEvent. "Why didn't you use PPostEvent? Don't you have a copy of Inside Mac IV? It's easy. Here's how..." The disclaimer above explains why I didn't use PPostEvent.

[Thank you, Matthew, for your update. Matthew is one of the

more conscientious writers, always thinking about his article, even after it has been submitted. Unfortunately, this letter came to me just as I was proofing the final proofs of the journal. -KC]

MacFortran/020 Math Library

Tatsuhito Koya

Evanston, IL

Being a student, I have been on a tight budget lately. I had to terminate most of my magazine subscriptions, but I have decided to keep *MacUser*, *Computer Shopper*, and *MacTutor*. These three magazines have been the main sources of information about what is happening in the world of personal computing—the world of the Macintosh, specifically.

I am not a computer science major; hence, it is difficult for me to find time to develop my programming skill, yet I do have opportunities to use computers for engineering computations. I do most of my computations on the main frame, but I would really like to use my Mac as well. I have MacFortran/020, but the problem is that I cannot find a math library for it anywhere. Moreover, Edit text editor is not one of the best program editors around. I would really appreciate it if you could recommend me a good math library (modifiable preferred) and a program editor.

[I called Absoft to check on the math library; unfortunately, they knew of none, but they seemed to like the idea; maybe our readers know of a library out there, or will create one. -KC]

[As for a text editor, I would suggest a new Fortran specific editor called **FREDITOR 1.0** from Battelle Pacific Northwest Laboratories, PO Box 999, Richland, WA 99352. Our good friend Bill Rausch wrote the manual. The editor was written in Lishgtspeed C and used the LS CAPPS Editor construction kit. -Ed]

IAC Driver

Lawrence D'Oliveiro

Hamilton, New Zealand

I read the articles on the IAC driver by Frank Alviani and Paul Snively with great interest. The Mac certainly needs a capability like this. However, a couple of questions.

First of all, why does the driver need to save multiple editions of a source extent? It only returns the latest one on any ReadData call anyway.

Secondly, it would be useful to allow desk accessories to take advantage of IAC. And of course, you don't need MultiFinder to run desk accessories. The check for the presence of MultiFinder is neat, but is it really necessary?

As always, keep up the good work, and let's have even more thought provoking articles in MacTutor! [We'll do our best. -ed]

[This letter gave me the excuse to call and talk to Frank Alviani. Frank, if you ever get the chance, is one of the nicest people to talk to, and I enjoyed the opportunity to call and see how things were going on his new house (not done as of this writing).

Frank explained to me that although the main purpose of the driver is to give the most recent extent to the clients, there are a couple reasons to keep the old extents. The first is to give the client the ability to call older extents. This could be used for perhaps a "hot" undo feature.

The second reason is useful for database type applications where each extent may want to be saved for future use. Lengthy

file operations may bring the client back in time to find himself several extent versions behind the current version.

Your second question was a good one, also. Desk accessories have the ability of file I/O. Also the driver itself makes ordinary calls. Therefore it is conceivable that the driver could be modified to work with desk accessories. The call to initialize the driver could say basically, "I don't care if MultiFinder is around." - KC]

Animated Watch Cursor Typo

Andy O'Brien

Keep up the great work on a fine magazine. Even the letters contain little tidbits of information. I especially enjoyed David Stoops' letter in the December '88 issue. After keying it in (and getting the proper LSP error message). I Realized that the InitWatch procedure in the WatchUtilities unit had a typo in it. GetCursor was being called as a procedure when we all know it is a function. Here is what the procedure should look like. Take care and continued success

```
porcedure InitWatch;
var
  watchcount : integer;
begin
  WatchList := acurHandle(GetResource('acur', 0));
  with WatchList^ do
    begin
      for watchcount := 1 to 8 do
        Watch[watchcount] :=
          GetCursor(HiWord(Longint(Watch[watchcount])));
        whichWatch := 0;
      end;
    end;
end;
```

Volume 5, Number 3

Watch Cursor Revisited

Jan Eugenides

Waterbury Center, VT

David Stoops' Animated Cursors from the December issue was interesting, but it had two deficiencies:

- 1) It was hard-coded for eight cursors, instead of obtaining the real number of cursors from the 'acur' resource,
- 2) It used SetUpA5 and RestoreA5, which should no longer be used.

I recoded the example in MPW C 3.0b1, with the modifications, and here is the result. I also changed to an 'acur' resource with an ID of 128, so as not to conflict with any system resources. The Rez source code for the watch 'acur' and the associated cursors are as follows:

```
#include <Types.h>
#include <Resources.h>
#include <QuickDraw.h>
#include <OSUtils.h>
#include <Retrace.h>
#include <Memory.h>
#include <ToolUtils.h>
#include <Strings.h>
#include <Errors.h>
```

```
typedef struct (
  short      numCurs; /*number of cursor "frames" */
```

```

short    whichCur; /*used as "frame" counter*/
CursorHandle  Curs[1]; /*can actually be any number of
cursors here*/
)acur;
typedef acur *acurPtr,**acurHandle;

static acurHandle  CurList;
static VBLTask      VBL;

pascal void AnimateWatch() /*the VBL routine*/
(
long    oldA5;

oldA5 = SetCurrentA5();
if(((*CurList)->whichCur >= (*CurList)->numCurs-1)
(*CurList)->whichCur = 0;
else
(*CurList)->whichCur ++;
SetCursor(((*CurList)->Curs[(*CurList)->whichCur]));
VBL.vblCount = 10;
SetA5(oldA5);
)

InitWatch()
(
short    curCount;

CurList = (acurHandle)GetResource('acur', 128);
for(curCount = 0; curCount < (*CurList)->numCurs; curCount++)
(*CurList)->Curs[curCount] =
GetCursor(HiWord((long)(*CurList)->Curs[curCount]));
(*CurList)->whichCur = 0;
)
StartWatch()
(
OSErr    err;

VBL.qType = vType;
VBL.vblAddr = (VBLProcPtr)AnimateWatch;
VBL.vblCount = 10;
VBL.vblPhase = 0;
err = VInstall((QElemPtr)&VBL);
)

StopWatch()
(
OSErr    err;

err = VRemove((QElemPtr)&VBL);
)

#include "types.r"

data 'acur' (128, locked, preload) (
$*00 08 00 00 01 00 00 00 01 01 00 00 01 02 00 00"
$*01 03 00 00 01 04 00 00 01 05 00 00 01 06 00 00"
$*01 07 00 00"
);

resource 'CURS' (263, locked, preload) (
$*3F 00 3F 00 3F 00 3F 00 40 80 80 40 90 40 88 60"
$*9C 60 80 40 80 40 40 80 3F 00 3F 00 3F 00 3F"
$*3F 00 3F 00 3F 00 3F 00 7F 80 FF C0 FF C0 FF C0"
$*FF C0 FF C0 FF C0 7F 80 3F 00 3F 00 3F 00 3F"
(8, 8)
);

resource 'CURS' (262, locked, preload) (
$*3F 00 3F 00 3F 00 3F 00 40 80 80 40 80 40 80 60"
$*BC 60 80 40 80 40 80 3F 00 3F 00 3F 00 3F 00 3F"
$*3F 00 3F 00 3F 00 3F 00 7F 80 FF C0 FF C0 FF C0"
$*FF C0 FF C0 FF C0 7F 80 3F 00 3F 00 3F 00 3F"
(8, 8)
);

resource 'CURS' (261, locked, preload) (
$*3F 00 3F 00 3F 00 3F 00 40 80 80 40 80 40 80 60"

```

```

$*9C 60 88 40 90 40 40 80 3F 00 3F 00 3F 00 3F"
$*3F 00 3F 00 3F 00 3F 00 7F 80 FF C0 FF C0 FF C0"
$*FF C0 FF C0 FF C0 7F 80 3F 00 3F 00 3F 00 3F"
(8, 8)
);

resource 'CURS' (260, locked, preload) (
$*3F 00 3F 00 3F 00 3F 00 40 80 80 40 80 40 80 60"
$*9C 60 84 40 84 40 40 80 3F 00 3F 00 3F 00 3F"
$*3F 00 3F 00 3F 00 3F 00 7F 80 FF C0 FF C0 FF C0"
$*FF C0 FF C0 FF C0 7F 80 3F 00 3F 00 3F 00 3F"
(8, 8)
);

resource 'CURS' (259, locked, preload) (
$*3F 00 3F 00 3F 00 3F 00 40 80 80 40 80 40 80 60"
$*9C 60 82 40 80 40 40 80 3F 00 3F 00 3F 00 3F"
$*3F 00 3F 00 3F 00 3F 00 7F 80 FF C0 FF C0 FF C0"
$*FF C0 FF C0 FF C0 7F 80 3F 00 3F 00 3F 00 3F"
(8, 8)
);

resource 'CURS' (258, locked, preload) (
$*3F 00 3F 00 3F 00 3F 00 40 80 80 40 80 40 80 60"
$*9F 60 80 40 80 40 40 80 3F 00 3F 00 3F 00 3F"
$*3F 00 3F 00 3F 00 3F 00 7F 80 FF C0 FF C0 FF C0"
$*FF C0 FF C0 FF C0 7F 80 3F 00 3F 00 3F 00 3F"
(8, 8)
);

resource 'CURS' (257, locked, preload) (
$*3F 00 3F 00 3F 00 3F 00 40 80 80 40 81 40 82 60"
$*9C 60 80 40 80 40 40 80 3F 00 3F 00 3F 00 3F"
$*3F 00 3F 00 3F 00 3F 00 7F 80 FF C0 FF C0 FF C0"
$*FF C0 FF C0 FF C0 7F 80 3F 00 3F 00 3F 00 3F"
(8, 8)
);

resource 'CURS' (256, locked, preload) (
$*3F 00 3F 00 3F 00 3F 00 40 80 84 40 84 40 84 60"
$*9C 60 80 40 80 40 40 80 3F 00 3F 00 3F 00 3F"
$*3F 00 3F 00 3F 00 3F 00 7F 80 FF C0 FF C0 FF C0"
$*FF C0 FF C0 FF C0 7F 80 3F 00 3F 00 3F 00 3F"
(8, 8)
);

```

Animated Cursors Again Erny Tontlinger Steinfurt, Luxembourg

Here are some comments to David Stoops' Animated Cursors from the December 1988 Letters Column.

There is an error in the procedure InitWatch. The cursor id should be replaced by the cursor handle in the loop this:

```
Watch[watchcount] :=
```

```
GetCursor(HiWord(longint(Watch[watchcount])));
```

The correct structure of a 'acur' resource is:

```

acur = record
frames: integer; {Number of frames(cursors) }
whichWatch : integer;
Watch : array[1..99] of CursorHandle
{ size depends on frames }
end;

```

Then instead of using a fixed number of eight cursors, use Watch^.frames, which happens to be eight for the Finder.

The 'acur' and 'CURS' resources should be locked, otherwise the VBL task runs into troubles with the memory Manager. The Finder uses the 'CURS' id 4 (classic watch cursor), which in the System file is not locked. This cursor is also in ROM, from the Mac Plus machines on, so it would be good to set RomMap-Insert to mapTrue before calling GetCursor.

SetUpA5 and RestoreA5 might not work as expected under MultiFinder. The value of A5 should be placed in a position, other than CurrentA5 in low memory globals, where the applica-

tion can find it from within the VBL task. See "Programmer's Guide to MultiFinder" (APDA# KMB017) the chapter "The A5 world/VBL tasks".

More Animated Watch Cursors

Rob Terrell
Chapel Hill, NC

In the December issue, you printed a letter that showed how to animate the watch cursor as a VBL task, which I thought was strange since a few issues back you printed another letter that gave a very good reason not to: a VBL task can go on and on while the application has crashed, and the poor user will be stuck thinking everything is okay.

A better way to implement the animated watch is through a unit that just takes a single call to increment the cursor. Support for this exists in TML Pascal II; for those not fortunate as to own TML, here's a unit that does essentially the same thing:

```
Unit WatchItSpin;
(*Make sure you have copied the 'acur' and 'CURS' *)
(* resources from the Finder into your application *)
(* ResEdit is quite handy for this *)
```

Interface

Uses MemTypes, QuickDraw, OSIntf, ToolIntf;

```
type
  aCur = record
    whichWatch : longint;
    watch : array[1..8] of CursHandle;
  end;
  acurPtr = ^ aCur;
  acurHandle = ^ acurPtr;
var
  currentWatch : integer;
  WatchList : acurHandle;
```

```
Procedure InitSpinner;
Procedure SpinWatch;
```

Implementation

```
Procedure InitSpinner;
var
  count : integer;
begin
  watchList := acurHandle(GetResource('acur',0));
  with watchList^ do begin
    for count := 1 to 8 do
      Watch[count] := GetCursor(HiWord(longint(
        Watch[count])));
      whichWatch := 0;
    end;
  end; (* InitSpinner *)

Procedure SpinWatch;
begin
  with WatchList^ do begin
    if whichWatch >= 8 then whichWatch := 1
    else whichWatch := whichWatch + 1;
    SetCursor(Watch[whichWatch]);
  end;
end; (* SpinWatch *)

end.
```

Most of this code was simply taken from David Stoops' letter. However, to use this, you call InitSpinner when you

initialize everything else at the beginning of your program; then you call SpinWatch each time through a loop or time-consuming process to increment it. Not quite as nice as having a VBL take care of it, but this way, if you crash, the hands will stop.

Volume 5, Number 4

FORTTRAN Math Libraries

Michael M. J. Tracy
Pittstown, NJ

I am writing in response to Tatsuhiro Koya's request (Feb 1989, Vol. 5 No. 2) for information on FORTRAN math libraries that will run on the Mac. There is an excellent book out called 'Numerical Recipes: The Art of Scientific Computing' by William H. Press, Brian P. Flannery, Saul A. Teukolsky & William T. Vetterling (Cambridge University Press, ISBN 0 521 30811 9). I refer to this book infinitely more often than 'Inside Macintosh' when writing scientific applications on the Mac (and it's cheaper too, about \$35 hard cover). The book provides the source code for over 200 subroutines. Its real value to the user, however, is in the eloquent and intelligent discussion of the principles underlying each algorithm, explaining the strengths and weaknesses, and usually providing alternative algorithms so that the cautious user can cross check results. I have learned a lot from this book. Listing from the contents page, the book covers; solution of linear algebraic equations; interpolation and extrapolation; integration of functions; evaluation of functions; special functions; random numbers; sorting; root finding and non-linear sets of equations; minimization and maximization of functions; eigensystems; fourier transform spectral methods; statistical description of data; modeling of data; integration of ordinary differential equations; two point boundary value problems; partial differential equations. There are two versions of the book. The original edition gave the subroutine listings in both FORTRAN and Pascal. By popular demand, a second version was released which presents the C source code listings (written somewhat from the perspective of a FORTRAN programmer who wishes to convert to C, and provides structures and functions for handling complex numbers). Macintosh compatible disks containing the source code are also available from the publishers.

All FORTRAN subroutines that I have used work well, and have compiled without any problems under both MacFortran and Language Systems Fortran. Some subroutines, however, need to have the 'SAVE' statement added at the beginning (as required by the ANSI 77 FORTRAN standard) if local variables need to be preserved between subroutine calls, such as in the random number generators.

Author Incentive Program Correction

Kirk Chase
Anaheim, CA

In the December '89 MacTutor, we made a correction on the Author Incentive Program initiated by Apple. Additional reimbursement is ONLY FOR APPLE EMPLOYEES. Those not employed by Apple DO NOT QUALIFY. We are sorry for any

confusion that may have resulted over this and hope this clears up any mistaken notions. If you are with Apple, and if you want more information, please contact Stacey Farmer in public relations, who is now in charge of the program, for more information.

Volume 5, Number 5

Aranda Revolutionizes Software Development

David E. Smith
Editor & Publisher

The microelectronics industry uses CAD software to develop complex IC's automatically through software. Now, thanks to a new software development utility called Aranda, software development will benefit from the same type of design automation that IC designers have enjoyed for several years.

Aranda is a compiler specific utility that opens and reads computer source code text files, parses the language syntax and builds an interactive symbolic flow chart representation that is linked by hypertext to the original program listing. Edit changes to either the graphical flow chart representation or the source code itself results in the other representation being updated automatically. Code can be cut and pasted in symbolic form, detail hidden or expanded, call links between modules called out and many other design tools used to facilitate and automate the software design process. But the important feature any programmer will love is that finally, software documentation and flow charts can be created after the fact and linked dynamically to the code so that the documentation and the code stay up to date at all times. Have a strange program you don't understand or didn't write, but are required to maintain or upgrade? No problem! Just run it through Aranda and the entire program structure, design and variables will be extracted and symbolically and graphically represented. The authors ran the entire MacApp source code through Aranda and out came a giant flow chart of the entire MacApp module tree in just a few seconds! Aranda can be used in a top-down software design environment from the beginning of the design cycle, as an expert system editor, or in bottoms-up redesign to extract the program structure and facilitate upgrades and maintenance of programs already written. Modules can be documented by appending interactive notes to the source code or equivalent flow charts, and the notes travel with the modules as they are cut and pasted into new programs. Now the dreary process of documenting and specifying the software design is automated and extracted from the source code itself, leaving the programmer free to do the fun stuff: writing code, not documentation.

Aranda is a product of Soft-SET Technologies of Vancouver in Canada and benefits from University work that allows the authors to generate new parsing engines for different compilers in a matter of days. MPW Pascal, Lightspeed Pascal and C versions are going into beta site testing now with Cobol, PL1 and C++ versions to follow. Pricing and product availability have not yet been finalized, but people who should know inside Apple are already drooling over the prospect of beta testing this product, something Apple normally never does for third party software.

For more information, contact Bill Campbell at (604) 734-1622 or AppleLink CDA 0380. MacTutor will be using this product as it becomes available, and we will have much more on this when we see the beta version, which should be shortly after Apple's Developer Conference, where Aranda will be introduced for the first time.

More Math Libraries

Bob Estes
Somerville, MA

I'm writing in response to T. Koya's request for information on Fortran math libraries in the February issue. The source code to all the routines in *Numerical Recipes: The Art of Scientific Computing* (Press, et al.) is available on Mac disks from Cambridge University Press, Order Dept., 510 North Avenue, New Rochelle, NY 10801 for \$39.95. The book, which has to be purchased separately of course, explains the routines and contains references to other work. You'd probably need the book to use the routines, but it's a valuable (700+ pages) reference anyway. The source code is available in Fortran, Pascal, and C. The C version has its own book and follows the quirks of the authors, who don't like C very much for scientific computing. You need to specify that you want the Mac disks. The code should compile with any compiler.

More on Numerical Library

C. H. Friend
Watford, Herts, England

I assume the inquiry from Tatsuhiro Koya in the MacTutor February 1989 Letters refers to replacements for the routines commonly supplied with most mainframe systems for performing mathematical tasks and special functions (usually NAG in the U.K.). Such a library is available in source code form as "Numerical Recipes" from Cambridge University Press.

Now that it is possible to get disks of these books in Macintosh format, here's a summary:

Numerical Recipes is a practical reference and textbook for anyone doing numerical analysis. The authors provide the techniques and programs needed, also advice on which methods should be used for solving problems, with practical emphasis on proven algorithms.

The reader is assumed to be mathematically literate and familiar with one of the languages used, but no prior experience with numerical analysis is required.

Books:

Numerical Recipes - the original in Fortran, with Pascal source in an appendix. Disks for either language available separately.

Numerical Recipes in C - the follow up to the above. Exactly the same coverage, with C source code. Disks of the book available.

Chapters in the books:

- 1) Programming style, conventions, accuracy.
- 2) Linear equations.
- 3) Interpolation and extrapolation.
- 4) Integration of functions.

- 5) Evaluation of functions.
- 6) Special functions (Bessel, probability, etc.).
- 7) Random numbers.
- 8) Sorting.
- 9) Root finding and nonlinear equations.
- 10) Minimization or maximization of functions.
- 11) Eigensystems.
- 12) Fourier transform spectral methods.
- 13) Statistical description of data.
- 14) Modeling of data.
- 15) Integration of ordinary differential equations.
- 16) Two-point boundary value problems.
- 17) Partial differential equations.

The original book was written in Fortran, with Pascal source as an appendix. The translation was done by machine, so some of the code might look odd to the experienced Pascal programmers. A project to improve this is under way, and should result in a separate Pascal edition later. Fortran source is F77 standard. Pascal is ISO standard, though the Mac disks have been altered slightly to declare "longint" where it matters and to show "include" files. If your Pascal uses units, the routines will all have to be amended accordingly, but this is not difficult.

The versions in the C book are a complete rewrite, and the introductory chapter includes some helpful discussion of C programming for numerical purposes. The C disks are in MSDOS format only, a Mac version is not currently listed.

There is also a parallel set of books and disks titled "Numerical Recipes Examples" which contain short demonstration programs and some data files showing the use of these routines. These are quite useful for testing compilers, etc.

Numerical Recipes - The Art of Scientific Programming: W. Press, B.P. Flannery, S. Teukolsky, W.T. Vetterling, Cambridge University Press 1986. 0-521-30811-9 818pp.

Numerical Recipes in C: W. Press, B.P. Flannery, S. Teukolsky, W.T. Vetterling, Cambridge University Press 1988. 0-521-35465-X 768pp.

Numerical Recipes Example Books: in UCSD-Pascal, 0-521-30956-5; Fortran-77, 0-521-31330-9; or C, 0-521-35746-2

Recipe Disks:

0-521-35466-8 (C) 5.25 in. MSDOS.

0-521-35501-X (Pascal) 3.5 in. Macintosh.

0-521-35469-2 (Fortran-77) 3.5 in. Macintosh.

Example Disks:

0-521-35467-6 (C) 5.25 in. MSDOS.

0-521-35502-8 (Pascal) 3.5 in. Macintosh.

0-521-35468-4 (Fortran-77) 3.5 in. Macintosh.

[Well, there seems to be enough information to get a handle on some math libraries. A special thanks to our readers who are always on top of things. -ed]

Volume 5, Number 6

Corrections
Don Koscheka

Arthur Young N0735

Joe, your article on adding windows to XCMDs was informative and, for the most part, well written. Your discussion of scroll bars needs clarification. You state that using ScrollRect to scroll a window is not the preferred approach because the window will not get redrawn until update time. This is not true. While Inside Macintosh does imply that you should pass the UpdateRgn of the current window to ScrollRect so that the vacated area gets accumulated into the window's invalrgn, this is not strictly necessary. Consider the following code fragment:

```

Procedure TrackScroll( theControl : controlHandle; thePart :
INTEGER );
VAR
    TempRgn      : Region;
    tempRect     : Rect;
    dh, dv       : INTEGER;
BEGIN
    (** determine what part of the control we're in... **)
    (** and set dh, dv accordingly **)
    ...

    (** now scroll the scrollable part of the window **)
    tempRect := theWindowsScrollablearea;
    TempRgn := newRgn;

    ScrollRect := ( tempRect, dh, dv, tempRgn );
    SetClip( tempRgn );

    (** call whatever routine draws window contents **)
    Redraw_The_Window;

    DisposeRgn( tempRgn );
    ...
END;
```

This code has the advantage of requiring only enough extra memory to store the vacated region. Moreover, the novice programmer need not write any special screen drawing code. Whatever code is used to draw at update time can be called directly to redraw the window at scroll time. Since the window is clipped to only the vacated region, this approach results in fast, smooth scrolling.

The more experienced user can query the region to determine exactly which rectangle to update so that the application draws only what will be displayed in the vacated rectangle. This is useful in applications like word processors where much of the update time is spent calculating where things need to go on the screen.

Another interesting consideration is the possibility of creating universal scroll bars. We accomplished this by having track scroll operate on a "scroll record" that keeps track of all appropriate information relating to a given scroll bar. One of the fields passed to the scroll record is the address of the routine to call in lieu of "Redraw_the_Window" in the above example. The result is quite satisfactory: one pair of scroll bars can be made to behave consistently regardless of the number of window variations in a given application.

Thanks for the bent ear. Good luck at Apple (as a six-year veteran, I can tell you that it's a great place to work...)

CDEF Corrections

**Kirk Chase
Anaheim, CA**

It was brought up to my attention of another way to include CDEFs for debugging purposes. James Plamondon, from Abacus Concepts, pointed out a better, more legal (?), way of setting the `ctrlDefProc` field of the control record.

```
const
  JMP_Instruction = $4EF9;

type
  JumpRec = record
    instruction : integer;
    function : Ptr;
  end; (JumpRec)

  JumpPtr = ^JumpRec;
  JumpHdl = ^JumpPtr;

var
  cdefJump : JumpHdl;
  ctrlHdl : ControlHandle;

begin ( Open_Example_Window)
  (allocate and initialize jump record)
  cdefJump := JumpHdl(NewHandle(sizeof(JumpRec)));
  cdefJump^.instruction := JMP_Instruction;
  cdefJump^.function := myCDEFProc;
  (this ends initializing the jump record)

  ...
  (get the control record)
  CtrlHdl := GetNewControl(ButtonID, MyWindow);

  ...
  (Set the ctrlDefProc)
  CtrlHdl^.ctrlDefProc := Handle(cdefJump);

  ...
end;
```

After diving into the tech notes, I also found what James already knew. James also mentions that the control definition proc should be locked by keeping it in code segment 1.

I appreciate James Plamondon help. The method I gave was essentially for prototyping controls. This made development time faster since you could bypass the Compile, ResEdit, Compile, Frustration loop. I hope Abacus realizes the asset James Plamondon is. Thanks A Bunch.

Volume 5, Number 8

**Font Tool
Fabien Samuel
Paris, France**

As always I read the March issue of your magazine with pleasure and interest. I would like to share a comment about Randy Leonard's article, "How To Write a Font Tool for MPW". To start with, all my congratulations for giving a very clear presentation of the matter. I think anybody who reads the article now has all the material to easily start writing his own MPW tools. I just have a comment about the use of `SetResLoad(false)`: as stated in IM, you should `SetResLoad(true)` as soon as possible after calling `SetResLoad(false)`, or you may be asking for trouble. Although there is no problem in Randy's program if it has a normal exit, it seems `SetResLoad(true)` will not be called when

the program terminates on an error condition, and this might be a problem.

It appears the only call in the program that might trigger Resource loading is `GetFNum`, so I would personally bracket this call with `SetResLoad(false)` and `SetResLoad(true)`. In that case `SetResLoad(true)` isn't even needed, because `GetFNum` automatically calls it before returning (IM I - 223).

**Resource Fork Problem
Paul Onstad
St. Paul, MN**

I enjoy your articles of Pascal code but have never come across an example which would lead me out of the problem that has stopped my application dead in its tracks.

1. I wish to maintain data in both forks of an applications HFS data file.

2. HFS provides only one Resource Manager call which considers the location of the file for subsequent resource calls, i.e., *OpenRFPPerm*.

3. However, when the user performs a "Save" or "Save as..." the existing file must be erased with *FSDelete*. Create then follows to create both forks of the file and the revised data is written to the data fork. The previous resource fork data is temporarily saved in memory by the application. The resource fork of the new file is now empty.

4. After the new data fork has been filled. A call to *OpenRFPPerm* is made to open the empty resource fork with the intention of using *AddResource* to place the memory-saved resource data back into the revised file. But, *OpenRFPPerm* returns an unsuccessful "-1" because of *ResError* -39 (logical eof encountered on the empty resource fork).

Without a valid *refNum* from *OpenRFPPerm* there appears to be no way to communicate back to the application the location (folder, etc.) in which the resource data should be restored on disk. The existing *refNum* was lost because the resource fork had to be closed before *FSDelete*—otherwise the file was "busy" and could not be deleted.

Additional (if of some use): *DetachResource* was used just before the file was erased in order to hold the resource data in memory while the actual resource (and data) file was deleted.

If you can provide any examples or information, it would greatly be appreciated.

**MIDI Problems
John Kaplan
Chicago, IL**

I have been following your articles in MacTutor for some time, but have only recently been able to try your MIDI routines. I was hoping you wouldn't mind a few comments and questions that came up for me as I worked.

First, I know your code was meant for the MDS assembler and LSP version 1, but I was using MPW for assembly, and had to use LSP version 2. (Version 1 cannot link MPW object files.) I found that LSP v2 would not accept an MPW object file with DC's in it so I declared all the variables as IMPORT's in the assembler, and as variables of corresponding types in the LSP

interface file. I didn't really understand why people objected to the use of A5 variables in the first place (since if your going to use MIDI in an application you're going to have to have space for your data somewhere anyway...) but that's not the point. The point is that after framing everything in PROC and ENDPROC directives for MPW and moving the blocks around to avoid forward references, the whole thing compiled and linked fine. The only other thing worth mentioning is that MPW assumes when a variable is imported that it is referenced through A5, so there are no explicit A5 references for the variables. (You probably know all this already, but I mentioned it just in case.)

The next snag I ran into was in testing the routines in LSP I wrote a little program that writes incoming MIDI information in the LSP text window. This was a big mistake, as LSP v2 seems to reserve the right to move your code around in memory as it calls graphic operations to update it's text window. After the ten or fifteen crashes it took me to figure this out, I rewrote my code so I made my own window, and drew the MIDI information in with drawstring. (And locked the CODE resource for the LSP project in ResEdit, just to be on the safe side.) Finally RxMIDIA worked like a charm.

Then I wrote a simple sequence in my test program that sends a note on, then a note off for the same note. This just doesn't work, and try as I might, I can't find what's wrong. I've been over my code 5 times and can't find a typing mistake, and I don't have the expertise in interrupts or serial chips to know exactly what's going on to debug it. I was thinking there might be another compatibility problem with LSP v2, or a problem with porting the assembly code from MDS to MPS that I don't know about. I was hoping you could provide some guidance. I am including all relevant code I used. Of course, if it's too much trouble to slog through all my code to find bugs, any hint you could give me would be greatly appreciated—even so basic as how to start figuring out what's wrong myself (chip manuals and the like?). (By the way, the hardware I used was a Mac Plus with both a Southworth Jambox 4 and an Assimitation MIDI Conductor. When I tried TxMIDIA, I got no light on the Jambox panel, no response from my synthesizer, and no message on my Yamaha MEP4 MIDI monitor—meaning either no recognizable message was sent or no message at all.)

Offending code:

```
...
outmidi := $0092;
txmidia(outmidi);
outmidi := $003C;
txmidia(outmidi);
outmidi := $0040;
txmidia(outmidi);

while button do
;
while not button do
;
while button do
;

outmidi := $0082;
txmidia(outmidi);
outmidi := $003C;
```

```
txmidia(outmidi);
outmidi := $0040;
txmidia(outmidi);
```

[Not being a MIDI expert myself, I'll leave this up to any MIDI experts out there. But Kirk Austin did an update to his MIDI routines for an article in May 1989. After a quick look, I did notice a couple differences between the code you sent in and Kirk's latest version. Kirk's latest MIDI code wasn't published due to space limitations. The code is on the source code disk (#44) if you wish to order it. —Ed.]

Powerful Math Libraries Available With Language Systems FORTRAN Drew Steils Herndon, VA

Language Systems Corporation announced that powerful math libraries from the Numerical Algorithms Group, Inc. (NAG) are now available for use with Language Systems FORTRAN. The mathematical library of scientific and engineering subroutines has not been available on the Macintosh until now.

"NAG is a well-known international producer of scientific and engineering subroutines for supercomputers, mainframes and high-end workstations," said Rich Norling, chairman of Language Systems Corporation. "Now, for the first time, Macintosh II users will be able to access the same algorithms used on supercomputers. NAG has made 172 of the most frequently used subroutines available in one package for the Macintosh II. This is a substantial portion of the NAG FORTRAN Library and provides broad coverage of the principal areas of mathematics and statistics.

Karl Knapp, Technical Manager of NAG, called the availability of the NAG Workstation Library for Language Systems FORTRAN on the Macintosh II "a significant development, particularly for engineers who find themselves coding mathematical formulas over and over again from scratch. The library should free them up to get on with their real work.

The NAG Workstation Library offers subroutines for a wide range of mathematical applications including matrix operations, optimization, linear algebra, time series analysis and many others. The software is of particular value in projects for aerospace, energy production, chemical manufacturing, electronic design, and other areas where advanced mathematics are required.

Language Systems FORTRAN was the first FORTRAN compiler to be based in the Macintosh Programmers Workshop (MPW). The compiler is full ANSI Standard FORTRAN 77 and has the most VAX-compatible extensions on the Macintosh. It has the full capabilities of the Standard Apple Numeric Environment (SANET[™]), Apple Computer's implementation of IEEE Standard 754 for Binary Floating-Point Arithmetic.

Language Systems FORTRAN Version 1.2.1 can be acquired from APDA. The NAG Workstation Library can be obtained from Numerical Algorithms Group, Inc. 1400 Opus Place Suite 200, Downers Grove, IL 60515 (312) 971-2337.

Volume 5, Number 9

Another Way To Post Events Matthew Snyder Fairfield, CA

In reference to Joel McNamara's article in the July issue, there's another way to post command key events, but you wouldn't know about it unless you'd read a somewhat obscure 1 page chapter in Inside Macintosh Vol. IV. PPostEvent posts an event just like PostEvent, but it "returns a pointer to the created queue entry."

Here's an example of its use. Like Joel, I needed to post an event and then fix the modifier field. This is an FKEY I wrote because too many times I found myself in a situation (like a dialog box) where ordinary paste did not work. Because the command and shift keys are down while the FKEY executes, just using PostEvent would result in all the posted events being command-shifted.

```
*include <EventMgr.h>
*include <OSUtil.h>

main() {
    long length, offset;
    int index;
    char **tHandle;
    char *tPtr;
    EvQEIPtr MyEventPtr;

    if (TEGetScrapLen() > 0) {
        ZeroScrap();
        TEToScrap();
    }

    tHandle = (char **) NewHandle(0);
    length = GetScrap(tHandle, 'TEXT', &offset);
    if (length > 0) {
        HLock(tHandle);
        tPtr = *tHandle;
        for (index = 0; index < length; index++) {
            PPostEvent((short) keyDown, (int) tPtr[index], &MyEventPtr);
            MyEventPtr->evtQModifiers = 0;
        }

        DisposHandle(tHandle);
    }
}
```

By the way, I very much liked Joel's sentiment that as programmers we can rise above the mere end user; we're not at the mercy of the "software gods." It is a good feeling.

Hex Conversion Temple M. Sarles Nashua, NH

I enjoyed inputting and getting to run the "ADB Demo" for LS pascal 2.0 from your volume 5 number 3 in March 1989. Programs like these give dabblers like me some necessary insight into interfacing with the toolbox and really making things work.

It was a bit of a disappointment to find in procedure QuerySystem the line "str1 := concat('System Version (must convert

to hex) = ', str2, chr(13));". (emphasis added) Well, I did. Enclosed find the listing of a test program with the function ToHex which accepts a long integer and returns a string representing the value in hexadecimal. I think the code with its comments pretty well explains itself. There were two surprises. I idly tried a negative number, only to get unexpected results. I forgot that negative numbers are represented in two's complement. An easy solution using the toolbox is presented in lines 21 & 22.

Not really solved is the need to make the string of hex digits a variable instead of a constant. If Hex_Char = '0123456789ABCDE' is declared as a constant then line 27 (... , Hex_Char [Work + 1] ...) gets a compile error of "Too many indices are being applied to a variable or expression." The reason for this is not clear, but making it a string variable works well enough.

To apply this to ADB Demo, add function ToHex to MyADBStuff in the IMPLEMENTATION section ahead of procedure QuerySystem. In QuerySystem, replace the lines:

```
NumToString(LongInt(theWorld.systemVersion), str2, chr(13));
str1 := concat('System Version (must convert to hex) = ',
str2, chr(13));
```

with:

```
str2 := ToHex(LongInt(theWorld.systemVersion)); (convert to
hex string)
str2 := concat(copy(str2, 1, length(str2) - 2),
'. ', copy(str2, length(str2) - 1, 2)); (put in period)
str1 := concat('System Version = ', str2, chr(13));
```

Appropriate credit should be given to Think Technology for their excellent implementation of Pascal and especially for their outstanding development environment (DEC & Apollo could take lessons).

```
program Test; (Just a place to try things out)
var
    I1: integer; (The value as an integer)
    I2: string; (What it looks like as a string)

function ToHex (Number_In: LongInt): string; (This
does the job)
var
    Accumulate: string; (I never work into return value)
    Hex_Char: string; (The set of Hexadecimal digits)
    Shift: Integer; (Offset into the LongInt)
    Suppress_Zero: boolean;
    (A switch to suppress leading zeroes)
    Work: LongInt; (The clipped byte from the Number_In)

begin
    Accumulate := ''; (Start with a blank string)
    Hex_Char := '0123456789ABCDEF';
    (Somehow this doesn't work as a Const)
    Suppress_Zero := True;
    (To start with suppress leading zeroes)

    (First take care of sign bit, dispose of two's complement )
    ( and then deal with short three bit high-order hex number )

    if BitTst(@Number_In, 0) then (The sign bit is set!)
    begin
        Accumulate := '-';
```

```

    Number_In := BitNot(Number_In);
    (Two's Complement cleared)
    Number_In := Number_In + $1
    (It's always one shy - back to DP101)
end;
Work := BitShift(Number_In, 1);
  (Bang to left - IM/I-472)
Work := BitShift(Work, -29);
  (Back to right - IM/I-472)
if Work > 0 then
  Accumulate := concat(Accumulate, Hex_Char[Work + 1]);

Shift := 4;          (Start with bit 5 - the 2nd digit)
repeat              (Now a loop for the other 7 digit)
  Work := BitShift(Number_In, shift);
  (Bang to left - IM/I-472)
  Work := BitShift(Work, -28);
  (Back to right - IM/I-472)
  if Suppress_Zero then
    if Work > 0 then (The beginning of what we want)
      begin
        Suppress_Zero := False;
        (A digit - No more leading zeroes)
        Accumulate := concat(Accumulate, Hex_Char[Work +
11)
      end
    else
      (Don't do anything!!)
    else
      Accumulate := concat(Accumulate, Hex_Char[Work + 1]);
      Shift := Shift + 4;      (Move over one byte)
    until Shift = 32;      (Bit 33 is out of bounds)

  ToHex := Accumulate (Put the result into return value)
end;
  (of function ToHex)

begin
  T1 := 1538;          (A famous number now-a-days)
  T2 := ToHex(LongInt(T1));      (Let's see what it is)

(The next line inserts the period needed to make
theWorld.systemVersion human readable)

  T2 := concat(copy(T2, 1, length(T2) - 2), '.', copy(T2,
length(T2) - 1, 2));
  writeln('T1 = ', T1, ' T2 = ', T2)
end.      (of program Test)

```

Fill Rectangle Tool Kevin Parichan Reedley, CA

In the article I wrote showing how to do the Spray Can and Paint Bucket tools I mentioned that there was a unit on GENie that implemented these and other tools. That unit is no longer available, so I thought that I would give away the source for the standard filled rectangle tool. I rewrote the code in C so as to show that I don't play favorites. With the routine given as is, you can easily modify it for ovals and round rectangles, filled or not. The routine uses the current grafport to get the fill pattern and the widths for the frame. Enjoy.

```
#define NULL 0L
```

```

Ptr NewBitMap(theBitMap, theRect)
  BitMap *theBitMap;
  Rect *theRect;
{
  theBitMap->rowBytes = ((theRect->right - theRect->left +
15) / 16) * 2;
  theBitMap->baseAddr = NewPtr(theBitMap->rowBytes * (theRect->bottom - theRect->top));

```

```

    theBitMap->bounds = *theRect;
    if (MemError() == noErr)
      return(theBitMap->baseAddr);
    return(NULL);
}

RectTool_Filled(where, workRect)
  Point where;
  Rect *workRect;
{
  GrafPtr workPort;
  BitMap workBits;
  PenState workState;
  BitMap oldBits, tempBits;
  Rect tempRect, theRect, aRect;
  Point newLoc, pivot;
  long position;

  GetPort(&workPort);
  workBits = workPort->portBits;

  if (NewBitMap(&oldBits, workRect) == NULL)
    return;
  if (NewBitMap(&tempBits, workRect) == NULL)
  {
    DisposPtr(oldBits.baseAddr);
    return;
  }

  CopyBits(&workBits, &oldBits, workRect, workRect, srcCopy, NULL);

  tempRect = *workRect;
  tempRect.right = tempRect.right + 1;
  tempRect.bottom = tempRect.bottom + 1;

  GetPenState(&workState);
  PenNormal();
  PenSize(workState.pnSize.h, workState.pnSize.v);

  pivot = newLoc = where;
  SetRect(&theRect, where.h, where.v, where.h, where.v);

  while (Button()) {
    GetMouse(&newLoc);
    position = PinRect(&tempRect, newLoc);
    newLoc.v = HiWord(position);
    newLoc.h = LoWord(position);
    if (!EqualPt(newLoc, where))
    {
      Pt2Rect(newLoc, pivot, &theRect);
      Pt2Rect(where, pivot, &aRect);
      UnionRect(&theRect, &aRect, &aRect);
      CopyBits(&oldBits, &tempBits, workRect, workRect, srcCopy, NULL);
      SetPortBits(&tempBits);
      FillRect(&theRect, workState.pnPat);
      FrameRect(&theRect);
      SetPortBits(&workBits);
      CopyBits(&tempBits, &workBits, &aRect, &aRect, srcCopy, NULL);
      where = newLoc;
    }
  }

  DisposPtr(tempBits.baseAddr);
  DisposPtr(oldBits.baseAddr);
  SetPenState(&workState);
}

```

Volume 5, Number 11

Apple Introduces Two New Macs
Apple Computer introduced the Macintosh Portable and the

Mac IIci at another fancy roll out at Universal Studios last week. In much the same fashion as the Mac II introduction, Apple invited the press and important industry representatives to hear John Sculley pitch his new toys and Jean-Louis Gassée attempt to assemble the portable from scratch. Fortunately it went together perfectly and we all cheered when he turned on the power and up came "Ta Da!" on the screen. The usual visual effects were there; laser beams, clouds of smoke, Macintoshes rising majestically out of the stage like granite monuments of progress. Everyone was properly impressed. There was even a fan club of school children wearing Macintosh shirts to stand up and cheer. I went with a very pessimistic attitude, especially about the portable, but I came away from it very upbeat about both machines. I think Apple has done a good job with these two products, even if the hoopla at the Universal Studios is wearing kind of thin.

The Macintosh Portable

If you were expecting a laptop, the portable will be a disappointment. My friend Jim Fitzsimmons of Mac America took one look at the portable and promptly went out and bought one of the new Toshiba Dynabook laptops. He wanted a more powerful version of a Radio Shack 100 laptop. The portable Macintosh may not be a laptop, but neither is it a transportable either. It is typical of many of the Japanese PC portables in size and weight. The portable weighs 16 pounds with a 40 meg hard disk, floppy disk and the lead-acid battery. The retail price of the portable including hard disk is \$6500. The developer price is about half that amount. The look of the machine is very similar to the high end Toshiba portables except that the Mac portable has a keypad/track ball option next to the keyboard, making it a bit wider than other PC portables.

Battery Is Design Key

The key to understanding the portable is the design decision made by someone at Apple that the portable must be a fully functional Macintosh on battery power alone. Which raises the question of how many portable PC users actually use their computers on battery power alone? Since the running time of most portable computers using NiCad batteries is only one or two hours, does anyone really use their computers on batteries? If that is the number one design criteria, then you need something other than NiCad to get more time from the machine. Hence the decision to go with a lead-acid battery. The Mac portable runs a true 6 to 12 hours of continuous use without re-charging and can be re-charged overnight (3 hours actually). Because the discharge rate is slow and even, sophisticated power monitoring and shut-down procedures can be implemented, guaranteeing the user never loses data. The Mac has a nine volt battery backed-up CMOS RAM as well as a power monitor desk accessory with suitable alarms. And the battery can be replaced with the power on without losing anything. The trade-off is size and weight. But you get what may be the first battery-powered full-blown PC in the world where you can actually do something useful all day before re-charging.

Another trade-off of the battery power requirement is that the components must be low power CMOS. However, Motorola does not make any of the 68000 family in CMOS except the

68000. Hence, the processor, and much of the power of the machine is decided for you. The portable has a CMOS version of the 68000 because that is the only processor available. To keep the machine from being a total wimp, Apple upped the clock speed to 16 MHz making it the same clock speed if not the same processor as the Mac II and the SE/30. The portable runs about twice as fast as an SE, which would be expected by doubling the clock speed, but not as fast as an SE/30 or Mac IIcx. The battery decision also determines memory configuration. Low power means CMOS static RAM chips. But CMOS is a bulky process of placing N-channel and P-channel transistors in series to form inverter pairs and is only available in very low densities. Hence Apple is only delivering 1 meg portables which can be expanded to 2 megs with a plug-in memory card. However, some third party vendors are said to be bringing 4 meg CMOS RAM cards to market for the portable which would give it a limit of 5 megs. From the appearance of the portable's motherboard, the chip density is only 256K by 1 bit per chip. Since the battery takes up some bulky space (4 inch height in the rear end), the necessary height of the case allows the addition of plug-in cards. Besides the memory board, you can plug in a 2400 baud modem card, upgraded ROMs, and a single processor direct slot, not compatible with any other processor direct slot in the Macintosh family (naturally). The back end of the portable includes the standard Macintosh connectors for Appletalk, SCSI hard disks and the ADB bus for the mouse. Since operation on batteries alone assumes you are not near an AC outlet and hence not near a desk, the portable has a track ball so you can use it on your lap. The logic of that decision in light of the fact we already concluded it was not a laptop, escapes me. Perhaps the motherboard had a fixed width that determined the case size, leaving room for something besides the keyboard, so the track ball was added. Actually, the track ball, being small in size, is much easier to use than other track balls for the Mac that I've tried. But I still prefer the mouse. I'm pretty sure the width of 15.25 inches and depth of 14.85 inches means it won't fit exactly on an airline tv tray, the only place I can think of where you might use it without a table top handy.

Other Features

Unlike other portable PC's, the Macintosh portable has a readable screen display. Each pixel is controlled by a single transistor, resulting in what is called an active matrix LCD display, as opposed to a passive display used by other systems. In effect, the display is a giant IC. There is no ghosting or cursor loss and the graphics response is "just as good as a Macintosh". This is probably the biggest achievement for Apple in the portable design. It really does not sacrifice anything in the display to be portable. The display is bright, easy to read, clear, sharp and fast enough for all Mac graphics applications. The trade-off is that the yields on the display are probably low, which could affect the number of machines Apple can make, and undoubtedly expensive, which explains the high price. One potential complaint is that the screen is not backlit, because of the power drain it would cause. But it shouldn't be needed as the screen is very bright and can be viewed at any angle clearly.

Besides the internal modem card option, an external video

adapter option is available that allows monochrome images to be sent to either Apple monitors or RGB, NTSC, PAL or SECAM standard televisions, videocassette recorders and project devices. This would make the portable ideal for on-the-road presentations, seminars and instruction settings. This could be the highest demand market for the portable.

An Alternative Portable

Suppose you made a few changes in the design criteria. What might you get differently? Suppose you assume that no one really bothers to use computers on battery power alone so you don't need lead-acid batteries. Now you can build a Grid type Macintosh portable. With AC power, you can use high density dynamic RAM chips and an 68030 processor. Instead of 32 individual static RAM chips, you can use simm modules. Instead of a specially designed low power 40 meg hard disk, you can use off-the-shelf 80 meg hard disks. If you assume a table top is always available, you don't need a track ball. This would cut the width from 15 inches to 12 inches and the rear height from 4 inches to 2.5 inches. The motherboard depth could shrink from higher density chips so the depth could go from 14 inches to 11 inches, making it just slightly bigger than a piece of paper. Now you have a true laptop at around ten pounds, that has the power of a Mac IIcx or even a IIci if you bump up the clock speed. All those in favor of a smaller, more powerful, *non-battery* laptop that can run MPW and MacApp, raise your hands!

The Mac IIci

The Mac IIci is a 25 mHz version of the Mac IIcx. In every other respect, it is exactly like the Mac IIcx with a couple of notable exceptions. The IIci includes a video controller on board that will drive any of Apple's monitors at a depth of 8 bits per pixel. The video RAM is taken from the system RAM. Apparently at pixel depths of 1, 2 or 4 bits, the on-board video is faster than going out over Nubus to an external video card, assuming you have at least 2 megs of memory. However, at 8 bits per pixel, the memory contention between the video processor and the 68030 makes the on-board video slower than going out to the Nubus. The video supports up to 256 colors or shades of gray on the Apple 13 inch color monitor (640 by 870 pixels).

In addition to the built-in video, the IIci has a slot for a RAM cache, although Apple will not announce a cache card until next year. The highest performance is achieved by using a Nubus video card with a cache card. Performance is said to be 45% higher than a Mac IIcx and higher still (up to 75%) using a cache card. The RAM in the IIci are 80 nanosecond chips as compared to 120 nanosecond chips in the Mac II. The retail price of the IIci is \$7,000 including a 40 meg hard disk. The developer price is about half of that. In comparison to the IIcx, the price differential for a 50% performance improvement is only \$800 and if you consider that includes a video card, the cost only \$300, a small price for a lot of increased horsepower. If you are going to buy a Mac II, there is no point in getting anything other than this new IIci.

New ROMs

The most notable improvement in the Mac IIci has nothing to do with hardware. The Mac IIci is the first Macintosh to get new 32-bit clean ROMs! The memory manager is 32-bit clean

and all the ROM code has been optimized for the 68030 processor. A lot of the system patches in the system file, including 32-bit quickdraw have been moved into ROM, increasing the ROM space from 256K to 512K in the Mac IIci. This baby is ready for system 7!

When Apple released the IIcx, the motherboard was completely re-worked to give a much more denser design than the Mac II. With the IIci, Apple has re-designed the motherboard with the addition of five new custom ICs. Once again, Apple has shown it's willingness to invest in top quality pc board design. The Mac IIci motherboard is a work of art. There are 6 very large custom IC designs including the video controller and the memory controller, plus the Motorola 68030, and 68882. There are 8 smaller custom chips, including the SWIM chip (revised woz machine disk interface) and sound chip. There are only 32 other chips on the board including the 4 ROM chips and all the Nubus sockets, RAM sockets, ROM socket and cache socket and still the board looks very uncluttered. The Mac IIci is by far the most powerful and impressive machine Apple has ever offered and certainly should give any PC configuration a run for it's money. Apple is stressing the total performance improvement of custom hardware design, new ROM coding, system software enhancements as well as the 25 mHz clock speed to promote the IIci as advanced as any of the high end PC designs. It is obviously aimed at the government and University research markets. It even includes a parity memory option, apparently needed to satisfy government purchase order guidelines that were designed to lock federal employees into IBM PC machines. With the advent of the superdrive that can read IBM formatted disks, federal employees can order Macintoshes directly now. In fact, rumor has it that the first two months of production for both the portable and the Mac IIci, 10,000 units each, has already been spoken for by Uncle Sam!

Apple's Comments

So what is Apple saying about it's product line now? At last John Sculley got off the presentation software kick he has pushed at the last two MacWorld Expos and returned to the theme that Apple builds computers for individuals that combine consistency, intuitiveness and ease of use for the way normal people work. This is the right message for Apple to be focused on. They have won the battle of convincing the world that Macintosh is easy to use, and graphically powerful. Now they should be pushing that message to milk it for all it's worth, instead of backing off looking for the next desktop revolution. In it's message to the press, Apple VP's seemed to be returning the message back to Apple's roots. Randy Battat, VP of product marketing stressed the consistency of the Macintosh interface across all it's platforms. The software doesn't change. The user interface doesn't change. Only the machine gets faster and more responsive or more portable and flexible. This goes to the heart of the IBM mess where IBM is investing millions into developing OS/2 and the Presentation Manager while ComputerLand dealers sell MS/DOS 3.3 machines running Windows. Is it any wonder IBM has reported third quarter results will be unexpectedly down?

Brodie Keast, product marketing director, stressed that

Apple pays attention to details because it designs for the individual, to put the individual in control, making computing tools transparent. This message strikes at the heart of the PC confusion of trying to configure and set up machines that require device drivers, screen drivers, choice of incompatible operating systems and windowing systems and make the whole mess work together with a variety of dissimilar software applications. Keast points out that for Macintosh, integration and detail results in a complete set of Apple and third-party tools that work well together. Consistency, compatibility and integrated networking were themes further stressed by Kirk Loevner, Don Casey and Mike Homer. The message that Apple is sending out is that now flexibility and choice can be added without sacrificing consistency and compatibility. This is the right message for Apple and we hope that Apple's national advertising in 1990 will concentrate on these strengths of the Macintosh design.

Adobe - Apple - Microsoft Connection

This year's Seybold Conference turned into a real shouting match as Apple and Microsoft proved they really do love each other after all and Adobe's John Warnock and Steve Job's tried to put the best foot forward at finding themselves the odd men out. Just a few months ago, Apple sold it's 16% interest in Adobe, makers of the Postscript standard. Anyone who was at the Developer Conference and saw System 7 understood why Apple had sold out. Apple wants to be in control of it's own destiny in regards to both it's imaging and printing technology. By paying royalties to Adobe for every LaserWriter that goes out the door, Apple was taking a financial beating. And Steve Jobs was giving them an emotional beating by pushing Display Postscript on his Next machine. Clearly Apple had to make a choice. Either it had to go with Display Postscript and lose more control to Adobe, or it had to do something about Quickdraw. It chose the latter. The new Apple font technology, dubbed Royal, brings screen outline fonts to the Macintosh product line. Now Quickdraw can print text just as good as Postscript. But it still can't rotate text or place text on an arbitrary path. Presumably Apple is hard at work on the next generation Quickdraw that will replace Postscript in functionality. The surprise move at Seybold was that Apple has licensed it's Royal outline font technology to Microsoft for inclusion in OS/2 and IBM officials have announced that Royal is both technically sound and will be present in future versions of OS/2, as reported by MacWeek. In return, Microsoft is licensing it's own Postscript clone to Apple, thus ending Apple's dependence on Adobe. It also eliminates 25% of Adobe's gross income and Adobe's stock fell dramatically from \$23 to \$15 on the day of the announcement.

What this means is that Apple, Microsoft and IBM have all agreed on an outline font technology that does not belong to Adobe. This leaves the Next machine out in the cold. It makes Macintosh and IBM PC's even more compatible in the area of typesetting and layout, which strengthens both companies. It could also lead to a graphics standard as well if Apple improves Quickdraw and eventually licenses that as well as it's font technology in the future. We don't expect that to happen, but it is clear that Apple and Microsoft are the big winners and Adobe and Next the big losers in this round of corporate hardball. Expect

Apple to further drive the nail in the coffin with a future line of cheap, Quickdraw laser printers, perhaps even a portable one for the Mac portable? Apple must continue to support Postscript of course, because Quickdraw is not presently an alternative and desktop design and typesetting has become Apple's main market advantage. But the writing is on the wall and Apple's direction is clear. There will be an improved Quickdraw to go with the outline font technology in future Macintosh system software releases that will eliminate the need for the slower Postscript interpreter.

Volume 5, Number 12

There IS an OOP Book
James Plamondon
San Mateo, CA

I have just received and read my copy of the September issue of MacTutor, and was pleased to discover that it was up to its usual high standards. Unfortunately, I was horrified to see one piece of misinformation presented — however unintentionally — as fact, when the truth could be of great use to your readership.

In the article MacOOPs!, by Jean-Denis Muys-Vasovic, on page 102, in the first paragraph, the author says that "Algorithms + Data Structures = Programs", by Niklaus Wirth, was the book that established the case for structured programming in the seventies. This is entirely correct.

Unfortunately, the author goes on to say that "This technology [OOP] has not yet got its "Algorithms + Data Structures = Programs"...", meaning that there is not yet any single book on OOP that explains its concepts, advantages, implementation, and so on. This statement has not been true for more than a year.

Everyone who has an interest in OOP should run — not walk — to the nearest computer bookstore and buy a copy of Bertrand Meyer's book, "Object-Oriented Software Construction", published in 1988 by Prentice Hall (about \$40). It is without a doubt the most comprehensive, concise, and yet readable book on Object-Oriented Design and Programming to date — and I have no doubt that it will soon be considered to be as important to the success of OOP as Wirth's book was to structured programming.

Just as Wirth's book explained and made the case for block structuring, strong typing, and functional decomposition, Meyer's book explains and makes the case for object-oriented design, based on considerations of correctness, robustness, extendibility, reusability, and compatibility. Meyer's book, like Wirth's, combines the rigorous precision of computer science with the practical aspects of software engineering. Wirth used his own, new language, Pascal, to present his ideas, because the other languages of the time were inadequate. Likewise, Meyer uses his own language, Eiffel, to present his ideas, because no other existing language so elegantly addresses both the theoretical and practical issues of object-oriented design.

Pascal went on to become one of the major computer languages of the seventies and eighties, due to its close association with structured programming. Its commercial use was hampered by the fact that it was designed to be a teaching aid, not

a language for software engineering. Eiffel, on the other hand, was designed from the first to be a practical tool for the development of commercial software. Its future is unlimited.

In every discipline, its practitioners struggle for years on the fringes of great breakthroughs — gravity, natural selection, relativity, plate tectonics — until some author comes along and puts it all together. That is what Wirth did for structured programming, and that is what Meyer has done for object-oriented programming. I was a geologist during the battles that led to the acceptance of plate tectonics, so I've learned to recognize a paradigm shift when I see one.

If knowledge is power, and the future of programming is OOP, then this book will give you the power to control the future of programming. Take the day off, grab the company credit card, go to the nearest computer bookstore, buy Meyer's book, and don't listen to another word about OOP until you're done reading it.

Stop reading this letter! Get moving! Run!
What, you're still here?

Virus?
George Patall
APO New York

I am new to Mac programming and an avid reader of MacTutor and need your help regarding Mac viruses and their eradication. My system (a Mac II with an 80MB HD) had a bad case of the "hangs" lasting for 15-20 minutes on bootup and when launching applications. I cleared out all the files and reinstalled them after examining each one with ResEdit. I installed Vaccine on my system and it seems to keep a vigilant eye on any attempts to add additional CODE resources. It warns me and asks for the usual permission to let it add or deny access to the CODE resource.

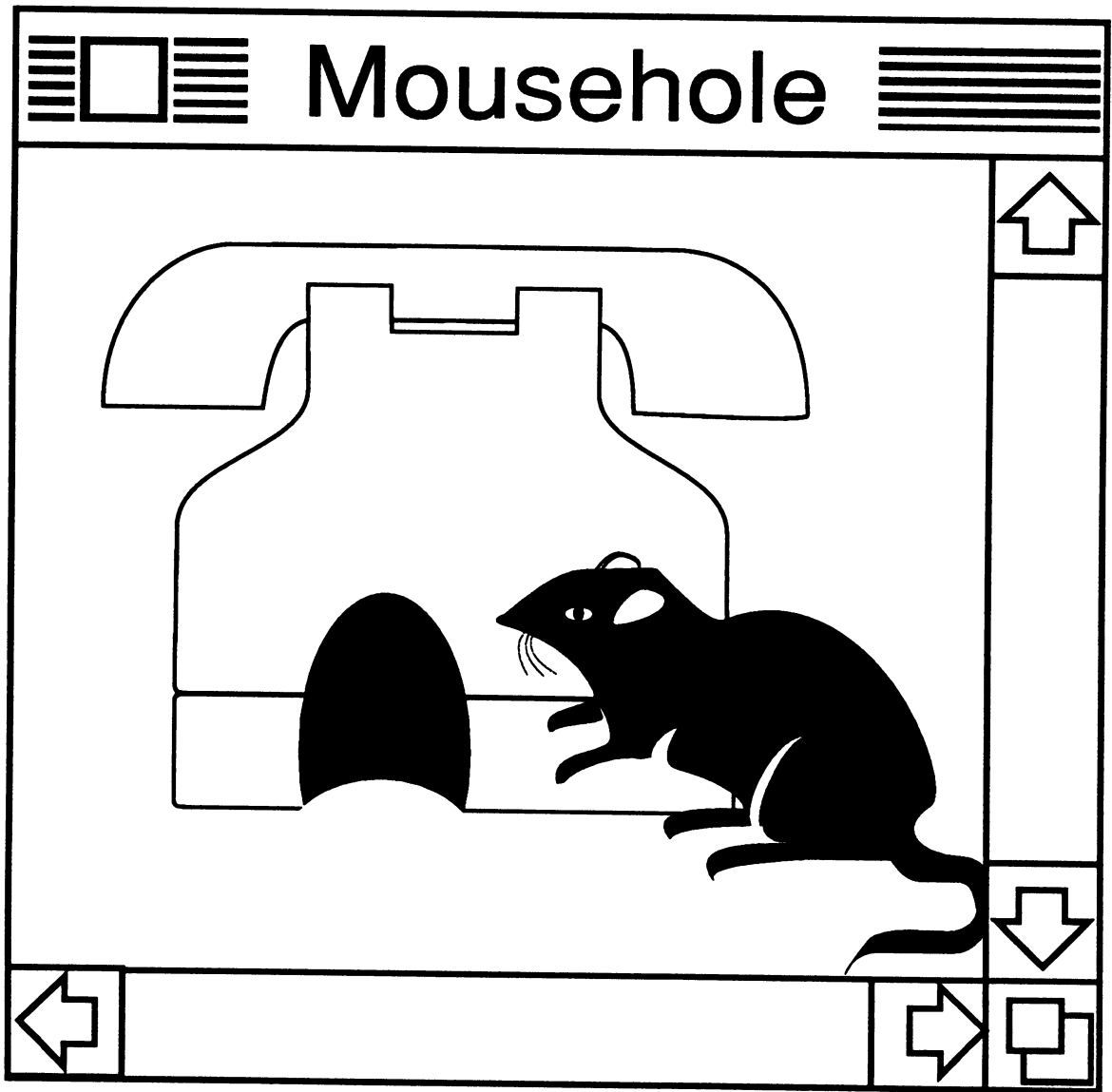
However running LS Pascal 2.0, it asks for permission with

the usual message for every source file I try to add to a project. I launched ResEdit and examined my System Folder for any telltale signs of Scores or nVIR viruses following Max Rochlin's instructions (Mousehole Report, MacTutor June 1988). None were found! However, the LS Pascal application's CODE id=0's eleventh word displays '038A' and the tenth displays '4EED'. According to MaxR this is a sure sign of Scores infection! But I could not locate any additional CODE segments corresponding to 038A which is two higher than the next, nor atpl id 128, data id -4001, inits 6, 10, 17, _Desktop and Scores files in the System Folder. I checked the master LS Pascal 2.0 disks I got from Think and they too display the same anomalies with 4EED and 038A. Is the LS Pascal infected, or is it normal code for LS? Do you think it is a new strain? It does not seem to spread to other applications so far. But I am at a loss as to how to combat this other than turning Vaccine off, which I don't want to do knowing if it is safe to run LS Pascal. I am stationed in Heidelberg Germany and the local Apple reps could not answer anything beyond the most elementary questions.

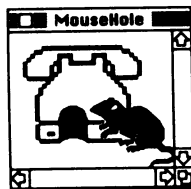
[LS Pascal adds code resources to its project file each time you compile, so naturally Vaccine will intercept the attempt to add a code type resource, like CODEs and CDEFs. My copy of LS Pascal also contained the same words in CODE 0. I have been using VIREX by HJC Software and Vaccine before without a hint of virus troubles. A quick call to Symantec said that the jump table is beyond that point and those are the words that should be there. So my opinion is that you don't have a virus. I would suggest your 'hangs' are due to INITs. One of them is likely to be giving you problems.

If you are worried about virus problems, may I suggest that VIREX or another commercial virus program. It is my understanding that Vaccine is not continually updated as other commercial programs. -ed]





Mousehole Report



*Rusty Hodge &
Larry Nedry
Mousehole BBS
(408) 738-5791*

Volume 5, Number 1

From: rusty (Rusty Tucker, Irvine, CA)
Subject: BitMap to Region

A while ago I thought that I saw something posted about code that will convert an Icon Bitmap into a region. Does any body know where I could find this, or have any examples or ideas on how to do this? Any help at all will be appreciated!!

From: emmayche (Mark Hartman, Fullerton, CA)
Subject: Re: BitMap to Region call

There exists a BitMapRgn() call from DTS. It's a (believe it or not) separately licensable product. We have a copy; as far as I can tell, I can only distribute it with Photon Paint.

What it does is to convert a BitMap (no PixMaps here) to a Region. FAST. What used to take 2 minutes now takes virtually no time. Great stuff. Give a call or drop some e-mail if you need any more info.

From: chenette (Phillip Chenette, Los Angeles, CA)
Subject: Double Clicks

This is probably a silly question, but I can't figure out how to detect a double-click of the mouse. I want to open up a dialog box when the user double-clicks on an object, but it seems there's no event message for this. What am I missing? (Using LSP 1.11) Thanks!

From: thecloud (Ken Mcleod, La Habra, CA)
Subject: Re: Double Clicks

You're right, there's no DbtClikEvt... you need to compare the time and location of a mouseUp event with those of the next mouseDown event, and if they're sufficiently close, the user double-clicked. Read Inside Macintosh Vol. 1, pp 255-56. If you're still having trouble, I can dig up some sample source code.

From: rdclark (Richard Clark, Tustin, CA)
Subject: Re: Double Clicks

The mouse isn't supposed to move more than 3 pixels between the 2 clicks, otherwise, it's considered a click-and-drag operation. Also, looking at the low memory global DoubleTime isn't such a good idea, as Apple is discouraging any direct access to the low memory globals. Use GetDbtTime() instead. Now for some code:

```
#define abs(x) ((x) >= 0 ? (x) : (0-x))
Boolean IsDoubleClick(where)
    Point where;
    (
        static Point oldLoc;
        static long oldTime = 0;
        long now = TickCount();
        Boolean idc;
        idc = FALSE;
        /* First, see if the 2 clicks are close enough in time */
        if ((now - oldTime) < GetDbtTime())
            /* They are, so see if they are close enough in space */
```

```
        if ((abs(where.h - oldLoc.h) <= 3) && (abs(where.v -
            oldLoc.v) <= 3))
            idc = TRUE;
        oldTime = now;
        oldLoc = where;
        return idc; )
```

From: emmayche (Mark Hartman, Fullerton, CA)
Subject: Re: Double Clicks and further checks

Richard is correct, as far as he goes; however, don't forget to check that the clicks are still within the same screen area (for example, the object on which your user will double-click to get that dialog). If he's real close to the edge on the first click, and just barely outside on the second click, it shouldn't be considered a double-click.

(I generally set a flag called checkDoubleClick in my main dispatch loop which checks the time and object, and let the routine which handles the object decide if it was indeed a double-click.)

From: dsa (Dave Stine, Saugus, CA)
Subject: MPW_RTExit routine

Just nailed a real weird one. I have a driver which patches the ExitToShell trap for applications which open a communications session in said driver. This is done because drivers running under MultiFinder don't receive a "goodBye kiss".

Well, it would seem that applications written in MPW C don't call the ExitToShell code in the normal manner; I think that in the _RTInit routine, they filch out the current routine in the ExitToShell trap address, stash this off in some parameter block, and it would appear that in _RTExit, they JSR (a0) to this entry point directly.

All of which means that if something patched ExitToShell *after* the _RTInit routine has run (which is caqllled before you main() entry is), you patch does not get invoked at program exit time. Can anyone confirm this for me, or am I just blowing smoke thru my hat?

From: rdclark (Richard Clark, Tustin, CA)
Subject: Re: MPW_RTExit routine

Yes, MPW C is doing something strange with ExitToShell(), and it seems that patching around the trap cannot be done. However, if you look at IM II-59, you'll see an "Assembly-language note" which implies that ExitToShell() calls Launch(). (And a quick test with MacsBug seems to confirm this. However, I couldn't see the "launch" execution under MultiFinder — I had to re-run the test under "flat" finder.) I dunno, this stuff sounds pretty dangerous.

From: dsa (Dave Stine, Saugus, CA)
Subject: Re: MPW_RTExit routine

Actually, the only reason I need to have the program come thru ExitToShell() is because MultiFlounder doesn't call drivers with the "goodBye Kiss" that the normal Finder does. So.... if some

application has a network session open thru my driver, and doesn't take care to close said session before exiting, there will be some grief Real Soon Afterwards.

So, the folks at Apple DTS told me to patch ExitToShell() to get around this "improvement" that MF made. So I did. And now, I find that yet another piece of Apple code has put a ditch in my path.

Yes, your suspicion about ExitToShell calling _Launch is correct — in the nominal Finder case. Under MF, things get much more hairy. When an application exits, MF must tear down the "sub-heap" that was created for the app to run in, and do some other funky things. If you want to see the beginnings of how MF launches an app, see Goldman's article in Aug. '88 Byte (of all places), wherein he describes some of what MF does. This was an especially nice article, full of info you simply can't get from DTS, even if you ask. (I did)

Anyway, I have already hopped around the problem for the time being. Thanks for your time.

From: adall (Alan Dall, Hampton, VA)

Subject: Re: MPW _RTExit routine

Rather than waiting for ExitToShell to be called for you, why don't you just call ExitToShell yourself when your program is done?

From: dsa (Dave Stine, Saugus, CA)

Subject: Re: MPW _RTExit routine

'Cuz my stuff is a driver, which is being used by the application which calls _RTExit, which doesn't call ExitToShell().

Calling ExitToShell() for a client application from a driver often result in much sadness.

From: rdclark (Richard Clark, Tustin, CA)

Subject: Re: MPW _RTExit routine

Whoa! If you look at the original post, it's the App which is supposed to call ExitToShell() — the driver is trying to intercept THAT call so it can close itself.

However, another ugly problem rears its head — many applications don't use ExitToShell(). Instead, they simply execute an RTS when done, which returns them to a place within the Launch() trap (which then returns them to the Finder or whatever). Time to go digging for some other way of shutting things down cleanly.

Volume 5, Number 2

From: thecloud (Ken Mcleod, La Habra, CA)

Subject: ScrollRect()

I'd like to be able to use ScrollRect to cause one bitmap to scroll down *into* a window, as the contents of that window are simultaneously scrolling down out of the visRgn. Trouble is, the update region gets filled with the port's bkPat (white) by ScrollRect, and this results in a "flash of white" before I can get the area redrawn. Can anyone point out a strategy for accomplishing this "smoothly"? I'm currently doing something like this:

```
for (i=1; i <= appropriate_multiple_of_inc; i++) {
    ScrollRect(&theRect, 0, inc, myRgnHandle);
    ScrollRect(&anotherRect, 0, inc, myRgnHandle);
    Draw_Stuff_In_anotherRect() }
```

but I'm sure I just need someone to kick me in the right direction!

From: rdclark (Richard Clark, Tustin, CA)

Subject: A kick in the right direction

Look at the articles by Scott Boyd ("The MacHax Group") in the past 2 years of MacTutor — they show how to do some pretty weird stuff with off-screen bitmaps and the like, including a trick to get rid of the "flash of white" when moving/updating a window.

From: lsr (Larry Rosenstein, Cupertino, CA)

Subject: Re: ScrollRect()

The best thing to do is to create an offscreen bitmap and use CopyBits to draw the bitmap on the screen in its proper position. If I understand your application correctly, you want to smoothly change from one image to the other. In that case, you would need to have the combined image offscreen, and use a series of CopyBits calls to draw the appropriate rectangle on the screen. (You would start out by copying the entire first image, then most of the first and part of the second, etc.)

CopyBits works the fastest when Quickdraw doesn't have to shift every scan line. If you can control the position of the window on the screen, then you should make sure that its global screen position is a multiple of 16 (or 32 on a Mac II).

There probably is a way to prevent ScrollRect from erasing things, but the resulting effect wouldn't be acceptable either. You would notice the delay between the ScrollRect and the redraw.

From: robert (Rob Anthony, Chicago, IL)

Subject: MacIntalk

Check out APDA (Apple Programmers and Developers Assoc., in Renton, WA). They had — in their 'Curiosity Shoppe' section — the MacinTalk Development Package v. 1.31. It has docs and tools for using MacinTalk (they say). I don't have it, so I can't give any personal experience, but the APDA folks should be able to answer your questions. Need to be an APDA member to order; about \$20 a year for that, but well worth it for the Mac programmer.

From: mwatts (Michael Watts, Santa Clara, CA)

Subject: limit on scroll bar value

Hi. I am new to the mousehole and have spent an informative session catching up. There is one question which I need help on if someone would be so kind.

In the app which I am writing, I have to deal with data which contains more than 32k points. I want to scroll through this data using a scroll bar, however the max value is an integer (opps). Even if I write my own control, in IM V I-333 the upper value for the CNTL resource is 2 bytes (not enough). I have kludged a fix by breaking the stream into 32k point sections, however this adds another control to my window and complexity to the program.

Does anyone know how to get around this problem, and if so how?

From: thotpolc (Bill Evans, Irvine, CA)

Subject: Re: limit on scroll bar value

"Doctor, it hurts when I do this."

"Don't do that...."

Your kludge to break the stream into 32k point sections is a

good start. I don't think the designers of the Mac interface and toolbox really anticipated someone scrolling more precisely than this. Notice, however, that the scroll bar is between two arrows, each of which points away from it. You could use these to scroll more precisely. To get the flavor of this, use any word processing program to create a document with more than 64k lines. (Probably short lines will make the whole thing a little easier.) Then scroll back and forth, using all parts of the scroll bar: the thumb (which is that funny white box you can slide back and forth with your... well, with your thumb), the gray parts of the bar itself, and the arrows at each end. In a well-tempered word processor, a single click in one of these arrows should move you up or down by one line; holding down the mouse in one of these arrows should, after a short while, start scanning through the document in small increments. Go thou and do likewise.— the Thought Police

From: rdclark (Richard Clark, Tustin, CA)

Subject: Re: limit on scroll bar value

There are a couple of ways to take care of your problem:

1) Redesign the software so that a single click on the down arrow moves you down by n points instead of 1 point (where n could be $1/4$ screen or something.) Some how, I don't think that'll be satisfactory for you. 2) Write a custom control definition that uses something larger than an integer (an unsigned integer, maybe?), then avoid using calls like "SetCtlValue" and "GetCtlValue". Fortunately, there's a set of options available to you in writing a custom control definition that allow you to get away without Set/GetCtlValue. The first problem involves dragging the "thumb" portion of the scroll bar. If you look at IM I-331 and I-332, you'll see that you can write a "drag", "position", and "thumb" routine that override the control manager (I suspect that the control manager uses the get/set control value calls.) These could read your "extended" position information and adjust the control accordingly. That leaves a different problem — what happens if the user clicks in one of the arrows?? You'll need to pass a message to your control to adjust itself by 1. The only reasonable way I can think of doing this is to make one of your control messages do double duty (you don't want to pick a new message number since Apple might decide to use that message someday, and that could hurt.) How about calling the control definition with a second (third, fourth...) "initCntl" message and a special value in the "param" parameter?? Your definition could recognize the special value (or the fact that you're already initialized) and adjust itself accordingly.

On second thought, why not just live with a scroll bar position which is an approximation of the real position?? After all, we won't be seeing 33K pixel by 33K pixel screens in the near future, so all your extra effort won't mean much on the display.

From: mwatts (Michael Watts, Santa Clara, CA)

Subject: Re: limit on scroll bar value

That is a good suggestion. I was fixating on using the scroll bar value as my numerical indicator on where I was in the data set. Your suggestion implies I have a position variable which is incremented by the scroll step parameter and then position the thumb relative to the size of the data set.—a very "thoughtful" piece of advice.

ps- I still consider this a kludge solution. there should be a way to have the value of the scroll bar be a long integer.

From: rdclark (Richard Clark, Tustin, CA)

Subject: Re: limit on scroll bar value

Actually, I like the thought police's answer MUCH better. (But I never could resist a good challenge!)

From: thotpolc (Bill Evans, Irvine, CA)

Subject: Re: limit on scroll bar value

rdclark (Richard Clark, Tustin, CA) writes:

On third thought, if you want exactly what you want without kludges, why not throw away the whole idea of part numbers and messages doing double duty and custom control definitions, and just do the whole durn thang using normal QuickDraw calls? I did this once with buttons which I wanted to have a special shape, and it was fun! If I ever use scroll bars, I'm going to want the thumb to be not square, but long enough to show what portion of the document is on the screen. That way, for example, if almost all of the document is on the screen and you're at the beginning of the document, you'll see a thumb which occupies almost all of the scroll bar, with a small gray area at the bottom. And if you click in that small area, you'll see a large thumb move a small distance (while the data on the screen is doing exactly the same thing), not a small thumb jump grotesquely from the top to the bottom of the scroll bar.

And if I ever get around to having this fun, you can bet I'll do it all without a formally defined control definition, thank you very much.

Does any of this give you the feeling that my favorite language is assembly? — the Thought Police

From: dhands (David Hands, Salem, OR)

Subject: MPW Print Interfaces

In the interface files of MPW Pascal there are two files, PrintTraps.p and MacPrint.p, that contain roughly the same calls. What's the difference and when should one be used over the other?

From: think (Think Technologies, Bedford, MA)

Subject: Re: MPW Print Interfaces

PrintTraps.p is the trap-based interface to the Print Manager; the traps are implemented in the Mac SE and Mac II ROMs, and in System 4.1 or later on all machines.

MacPrint.p is the glue-based interface, and will work on all Macs, on all versions of the System software.

Volume 5, Number 3

From: pmilltch (Peter Milltch, Laurel, MD)

Subject: RESEDIT

I was experimenting with RESEDIT and using it to look at some applications. I found that it will not open the CODE 1 resource in WORD or MACPROJECT 2. On other applications, no problem. I do not get a cannot do message or any indication of why it doesn't work - it just doesn't. Any theories on why this might happen.

From: emmayche (Mark Hartman, Fullerton, CA)

ResEdit won't open resources that are larger than a certain

value. What that value is, I don't know, but it's very obstinate about it. *[The CODE 1 resource in Word is probably the heart of the program itself, some 300K or more? Why would you want to open this anyway? ResEdit is not a disassembler. -Ed]*

From: thecloud (Ken Mcleod, La Habra, CA)

Subject: Re: INIT/cdev communication

I'm having a problem with tracking a scroll bar in my cdev's DITL. The Control Panel, attempting to be helpful, calls TrackControl() BEFORE sending a 'hitDev' message, so any attempt to use the no parameters, so naturally, a click in the thumb will kill the cdev instantly). I suspect the solution requires trickery (perhaps changing the "itemType" of its dialog item?). Has anyone else run into this?

From: lsr (Larry Rosenstein, Cupertino, CA)

Subject: Re: INIT/cdev communication

You may need to write a custom CDEF to replace the standard scroll bar CDEF. This CDEF can simply call the original CDEF for everything except the autoTrack message. It is whether you are dealing with a indicator or not.

From: thecloud (Ken Mcleod, La Habra, CA)

Subject: Re: INIT/cdev communication

Writing a custom CDEF within my code might work; it does seem like a lot of unnecessary coding, though :-('hierDA' is an example of a cdev that includes a 'fully functional' scroll bar in its DITL... the standard scroll bar proc is specified in the resource file, but perhaps he does some runtime twiddling... hard to say. I'll experiment. I enter a loop which seizes event-handling control away from the Control Panel. Each time through the loop, I check EventAvail() to see if there's an event (other than a null event) needing to be handled; if there's a mouseDown, and it's in the scroll bar's itemrect, I call my DoScroll() function to handle it. All other events (including mouseDowns NOT in the scroll bar rect) cause the loop to be exited, returning control back to the Control Panel...and then back to me, on the next null event.

Volume 5, Number 4

From: dbleber (Douglas Bleber, Alta Loma, CA)

Subject: PICT display

HELP!! I am currently using a Mac II w/a Supramac 19" color monitor. I am programming under MPW C 2.0.2. I have been attempting to convert the PICT file Pascal tech-note examples into work-able "C" code. Unfortunately, I have not been successful. I would be interested in any "C" source that reads and displays a PICT-2 file. Any help would be appreciated. Thanks.

From: emmayche (Mark Hartman, Fullerton, CA)

Subject: Re: PICT display

Here's some Lightspeed C code that I've written which does what I think you want to do. This also takes care of picture spooling. I'm not sure how much will be applicable to MPW C.

```
int thePath;
pascal void GetPICTData(dataPtr, byteCount)
Ptr dataPtr;
int byteCount;
{
```

```
    long longCount;
    longCount = byteCount;
    fileErr = FSRead(thePath, &longCount, dataPtr);
}
ReadPICTFile(path, window)
int path;
GrafPtr window;
{
    QDProcsPtr savedProcs;
    CQDProcs myProcs;
    PicHandle myPicture;
    long longCount, myEOF, filePos;
    Rect r;
    thePath = path;
    SetStdCProcs(&myProcs);
    myProcs.getPicProc = (Ptr)&GetPICTData;
    savedProcs = window->grafProcs;
    window->grafProcs = (QDProcsPtr)&myProcs;
    SetPort(window);
    myPicture = (PicHandle)NewHandle(sizeof(Picture));
    if (fileErr=GetEOF(path, &myEOF))
        return FALSE;
    if (fileErr=SetFPos(path, fsFromStart, 512))
        return FALSE;
    longCount = sizeof(Picture);
    if (fileErr=FSRead(path, &longCount, (Ptr)*myPicture))
        return FALSE;
    r = (*myPicture)->picFrame;
    DrawPicture(myPicture, &r);
    if (fileErr=GetFPos(path, &filePos))
        return FALSE;
    KillPicture(myPicture);
    window->grafProcs = savedProcs;
    if (filePos != myEOF)
        return FALSE;
    return TRUE;
}
```

Note that there are some casts to QDProcsPtr when you'd think it should really be CQDProcsPtr; don't worry about these, they work fine and leave the flexibility to return to black-and-white should you ever be crazy enough to do so. "fileErr" is a global error code declared in a .h file.

Hope this helps - please let me know. *[Another good source is Dave Wilson's Color Paint Example program in his Macintosh II Programming Class Notes, available from the MacTutor Mail Order Store. -ED]*

From: andyv (Andrew Voelker, Long Beach, CA)

Subject: Mac II bus error

I wrote a program that works fine on my SE, but it bombs with a bus error on a Mac II. What does a bus error mean on a Mac II?

From: emmayche (Mark Hartman, Fullerton, CA)

Subject: Re: Mac II bus error

It very much depends upon what you're doing. Generally, a bus error means that somebody (generally the CPU) has asserted an address and nobody lives there (actually, nobody has responded within the time limit of the bus).

If you can figure out about where you're bombing and post some of the code around that point, it would be helpful in the diagnosis.

From: lsr (Larry Rosenstein, Cupertino, CA)

Subject: Re: Mac II bus error

You can never get a bus error on the Mac SE (or Plus) because of the way the hardware is designed. Your program is

still accessing some strange error, but the hardware is mapping that access back to some valid memory address. This means you could be stomping on random memory locations on the SE.

From: dbleber (Douglas Bieber, Alta Loma, CA)
Subject: PICT display again...

The code that emmayche provided worked great! However, I am having two problems:

1) A color picture created by MacDraw II will not appear in color using the aforementioned code. Perhaps further "tweaks" are necessary.

2) A color picture created by Pixel Paint will appear in color. However, the update region (the area of the picture window covered by the SFGGetFile dialog) is updated in black & white. I am using CopyBits in my update function. I have retrieved the picture window using GetNewCWindow, and I have defined an off screen PixMap for use with the CopyBits routine. Unfortunately, when I initialize the system, the pixSize variable has a value of zero. I suppose I need to do further "initializations" not previously required for B&W. But what??

Any code and/or suggestions would be appreciated. The February 1988 issue of MacTutor has an excellent article entitled "Macintosh II: Color Screen Dump FKEY". The use of PixMaps is demonstrated in a Pascal DA. Unfortunately, no update procedures are performed.

From: emmayche (Mark Hartman, Fullerton, CA)
Subject: Re: PICT display again

It is unclear to me how your program is handling the update event that it will receive after the SFGGetFile dialog clears away. If you are updating from an off-screen PixMap, you should be sure that the offscreen port is opened with the proper depth, and use the OpenCPort command for it.

I open my palette window first with a GetCWindow call, then steal the depth of that window for document windows. Works fine unless someone makes the screen shallower on you (deeper, we can handle). This is tricky, which is one of the reasons Pixel Paint *requires* 256-color mode to be set. (My program, Photon Paint, is more intelligent, so it'll operate in 16- or 4-color mode. It'll do 2-color mode, too, but that gets boring.)

If this doesn't give you enough help, let me know and maybe we can do something about it off-line.

From: andyv (Andrew Voelker, Long Beach, CA)
Subject: my bus error

I get my bus error on the Mac II in the first instruction of this function:

```
void DrawCard(thePile, cardNumber)
struct pile*thePile;
char cardNumber;
{
    struct cardtheCard;
    theCard = *(thePile->cards[cardNumber]);
}
```

the following are global variables:

```
struct card {
    Rectrect
    char suit, value;
    Boolean facingUp;
} deck[NUM_OF_CARDS];
struct pile {
    RectbaseRect;
```

```
char numInPile;
struct Card*cards[MAX_CARDS_IN_PILES];
} playPile[NUM_OF_PLAY_PILES], accumPile[NUM_OF_ACCUM_PILES],
drawPile, discardPile;
```

It doesn't bomb on a Mac SE.

From: emmayche (Mark Hartman, Fullerton, CA)
Subject: Re: my bus error

Try changing:

```
theCard = *(thePile->cards[cardNumber]);
```

to:

```
theCard = *(thePile->cards[(int)cardNumber]);
```

I think that the byte-wide subscript may be killing you here. It's probably got some junk in the upper byte(s) which are pointing way-the-heck off somewhere. This is a wild guess - everything else looks fine.

Volume 5, Number 5

From: sstaton (Steven R. Staton, Dallas, TX)
From: rustyt (Rusty Tucker, Irvine, CA)
Subject: File Comments

I'm trying to devise a way to implement "File commenting" ala the Finder. One of my requirements is that the comment is accessible even if the Finder is running and has control of the Desktop file, it is not necessary (although desirable) that the Finder display this comment in it's "Get Info" Dialog.

An approach that I'm currently considering is to create a "FCMT" resource and add it to the file's resource fork (both APPL and doc). One of the benefits of this strategy is that the comment would be passed along when the file was copied or transferred via Modem.

My questions are:

Will this crash any of the existing Applications out there?

Will the Finder recognize this RSRC when it updates the Desktop file? Is this completely immoral?

From: alex (Alex Curylo, Ottawa, ON)
Subject: Re: File Comments

I think I've heard that messing around with an application's resource file just isn't a cool sort of thing to do.

F'rinstance, any application that's had 'JumpStart' move its resource map to the beginning of the resource fork will go belly up after you do this FCMT thing with it. Anybody in general who expects their resource map not to change will be in trouble. I don't know any specific applications that brain-dead, but I'm sure you'll find them for us.

From: rustyt (Rusty Tucker, Irvine, CA)
Subject: Re File Comments

Alex, I was thinking about your comment about "Jump Started" App's crashing after having their RSRC fork modified. My original reaction was very similar. Somehow it just doesn't seem Kosher to modify a file that you don't "own".

But isn't that what a large number of people do with ResEdit etc.? Modify resource forks to customize an application? Isn't

that part of the greater purpose of the Resource Manager?

As a little background, the FCMT resource is the STR resource type used by the Finder to manage the info string in the get info comment. It looks as if it takes the File number created by the File system and uses it as the res ID#. I'll test that this week and let you know.

Anyway I got to thinking about the other resources modified by the Finder and started poking around in my Desktop file for a SIZE resource that would correspond to one I modified in an APPL through the "Get Info" dialog. No SIZE RSRC's in the desktop file. Hmm, I looked in my APPL file and found a SIZE RSRC=0 put there by the Finder.

Maybe somebody from Apple could tell us if the SIZE resource is a special case, or if it's OK to add resources to files not owned by that Application.

From: rdclark (Richard Clark, Tustin, CA)
Subject: HyperCard errors

I may be proven wrong, but the only mention I've ever seen of the errors is in Goodman's Developer's Guide, and he waves them off with the comment "one of HyperCard's many internal consistency checks failed" and proceeds to note that your stack is now damaged beyond repair. No explanations. No hope of recovery. (almost makes one want to take up knitting instead.)

From: jflischer (Jeff Fischer, Carbon Canyon, CA)
Subject: Hypercard errors

Yeah, I've got that book. Big help, right? Why couldn't he have supplied them in an appendix with some sort of "at your own risk" caveat? Geez, I've got a 4.5 MEGABYTE stack that I'd do anything to recover, if only I could figure out what error 5544 is trying to tell me! Is the structure of a stack published? I mean, how it's all linked together internally? I have a PD program called "Stack Detective" that lists all the parts and how much room they occupy, I wonder where he got his info? Oh, well...I'll stop rambling—anyone out there with the low-down on the HyperCard error meanings and/or stack structure documentation PLEASE come forward!

From: ms (Mike Stelner, Sierra Vista, AZ)
Subject: Re: Hypercard errors

With that large a stack, this might not be workable, but... If you change the file type to TEXT, you can then read the entire document with a word processor. All the scripts and the text that appear in fields (but not graphics text) will be recoverable. I think that object names will be identified, but that probably won't be of much help though.

From: jflischer (Jeff Fischer, Carbon Canyon, CA)
Subject: Re: Hypercard errors

Thanks, Mike. You're right, this stack is too large for that solution. It is built largely of imported text, and I can rebuild it that way (since I saved the source), but I was trying for an even easier way out. I've been through the stack pretty thoroughly with FEdit, and discovered some pretty interesting things, but couldn't quite get enough of a handle on it to completely recover

it.

I now have in my possession Apple's tech notes on HyperCard (all 3!). The one titled "HyperCard File Format" says "HyperCard's file format is proprietary and will not be documented." Sigh. Oh, well, back to the quest!

From: ms (Mike Stelner, Sierra Vista, AZ)
Subject: Re: Hypercard errors

Well, Jeff, maybe as a side effect of your problems, you could put together those pretty interesting things you discovered about HC and publish an article in MacTutor (at least you would get a few \$\$ for your troubles).

From: jflischer (Jeff Fischer, Carbon Canyon, CA)
Subject: Re: Hypercard errors

Hmmm. Now there's an idea. Maybe I'll try to wake up the guy that wrote Stack Detective (he hasn't cashed my shareware check in two months—a bad sign). Now THERE's a guy who knows the guts of HC. I wonder where he got his info?

From: alex (Alex Curylo, Ottawa, ON)
Subject: DeskHook

Has anyone out there ever used the global 'DeskHook'?

It looks like you could do some neat things with it, but the Multifinder programming docs say it's not supported anymore. Comments, anyone?

From: chally (Mark Chally, West Covina, CA)
Subject: DeskHook

Yeah...I used to use it—until I was told it would break..it was great. Using it, you could send the address of a routine that polls the serial port and sends checksums in response, etc...just as long as it doesn't draw on the screen or change the menu items. There was also a dialogHook. I use neither anymore because I'm told they'll break. I will instead attempt to do a VBL task to do what I need to do 100% of the time instead of "patching it in" as I've described.

From: ericllm (Eric Lim, Flushing, NY)
Subject: MPW Resume Function

MPW Users:

Add the following lines around your DirectoryMenu and BuildMenu code in the UserStartup file:

```
*****
if 'exists -f "(ShellDirectory)MPW.SuspendState"'
    "(ShellDirectory)Resume"
else
    # place your regular DirectoryMenu and
    # BuildMenu code here
    DirectoryMenu '(Files -d -i "(MPW)"~Examples~ 6
    || Set Status 0) 3 Dev:Null' 'Directory'
    BuildMenu
end
*****
```

NOTE: Substitute the '~' with Option-X.

Append the following line to the Quit file:

```
*****
"(ShellDirectory)"suspend
*****
```

Now you have a real 'Resume' function not only when you

launch an application, but also when you 'Quit' from MPW. If you want to start with a clean desktop, just trash the MPW.SuspendState file and launch MPW Shell. Enjoy!

From: kdc (Kevin Connery, Bellflower, CA)
Subject: Novice questions re: C

I realize this is probably pretty basic for most of you, but everywhere else I asked it was too advanced, so please bear with me.

I'm just starting to program on the Mac, have decided that C is the language to use [don't ask me why!], and have discovered a number of obstacles in the path of smooth coding....

Are there any good books available on using the toolbox with C? I'd prefer a recommendation from someone who's using one, but barring that, ANY titles/authors would be appreciated.

And recommendations as to development systems also. I've heard good things about LSC, MPW, and AztecC, and have a fairly limited C background, with even more limited Mac coding background [none]. Suggestions? My goal is NOT to make the next PageMaker-level application, but I want to be able to effectively use the power the machine is capable of. Current system is a 1 meg Mac II, so anything which works here is possible...Many thanks in advance!

From: thecloud (Ken Mcleod, La Habra, CA)
Subject: Re: Novice questions re: C

Kevin, you can't go wrong with LSC... less than \$100 mail-order, and worth every penny. Since you're "not going to be making the next PageMaker," you can probably even use the source-level debugger on your 1 meg machine! (There's a trick to it, but it's not difficult.) Even without the source-level debugger, LSC is far and away the best C environment for the novice programmer (in much the same way that Lightspeed Pascal used to be the best Pascal environment). As for books on using the Mac toolbox with C, get your hands on "Using the Macintosh Toolbox with C" by Huxham, Burnard, and Takatsuka; published by Sybex. The original edition used 'Mac C' in examples, but I've heard there's an updated version in which all the examples are now in LightspeedC. Better yet, download all the sample C source code you can, and play with it... 'c' how it works. Good luck. *[Buy the back issues of MacTutor, published in our yearly Complete MacTutor books. -Ed]*

From: tycho (Donald Tycholls, Westminster, CA)
Subject: Re: Novice questions re: C

I have been using C on the Mac for the last 4 years, and am still somewhat of a novice with respect to the Mac Toolbox. I have used Manx Aztec C (currently with the new source level debugger) for 4 years, MPW C for 2 years, and LCS 3.0 for about 2 months. LSC is the winner for ease of use, speed of compilation/link time, and ease of debugging. In short, it is the most productive tool for me. Of course I had to upgrade to 2 MB and MultiFinder to run the debugger, but I have made more progress on my "project" in the last two months that I have had LSC 3.0 than I did in the last two years using Manx and MPW. Of course, both of the other compilers have their strengths; they are better

sued to larger, professional programming projects (especially MPW). I just program on the weekends and on an occasional holiday while the family is downstairs chatting around the turkey in the oven so I probably qualify as a bonafide amateur MacHacker. LSC is the best as far as I'm concerned, and for the first time the Mac has a C compiler which is better than MicroSoft C 5.0 on the PC (which I use at work)! (Actually, MPW and SADE may be better in some ways. I'll never know because I cannot justify buying a MAC IIx + 4 MB RAM just to get the same capabilities that LSC 3.0 provides). The latest rumor is that LSC will also support Object C soon and C++ later, so I am content with LSC at present.

As far as toolbox programming, I recommend "Using the Macintosh TOOLBOX with C" by Takatsuka, Huxham, Burnard (Sybex). For getting an understanding of The Mac internal concepts and debugging, I highly recommend both of Scott Knaster's books (How to Write Macintosh Software is the first and I don't remember the second since I just browsed it at B Dalton's).

Volume 5, Number 6

From: Rguerra
Re: FINDER QUESTION

Does anyone have any idea as to how the Finder places icons in its windows? I know about the FinderInfo fields returned by the PBGetCatInfo call, but what coordinate system is being used? The Finder does a SetOrigin on each of its windows so the coordinate system is somewhat odd. The POINT coordinates in the FInfo record very often will NOT correspond to a point in the visible window even though I can see the icon. Also, does anyone have any information on how the Finder keeps track of the files in each of the windows and how it determines which icons are selected and are to be opened by a double-click or an "Open" menu. I know this is all undocumented but I'd very much like to know how this works. Thanks!

From: Spud
Re: MultiFinder bug???

I have noticed that when calling an alert in the background under MultiFinder, the alert does not re-draw correctly. I have noticed this in many commercial apps, as well as my own. Here's the sequence of events:

- 1) Write an app that waits 5 seconds, and then puts up an alert. ProtoTyper, no problem.
- 2) Run the app. Within those 5 seconds, switch into another application whose window covers all others.
- 3) Okay, 5 seconds are up, so switch back into your original program. Notice that the icon and the alert's bold item are not drawn.

Is this a bug in MultiFinder, or am I just seeing things? If it is a bug, then how can I program around such a dilemma?

Thanx in advance,
-Spud

From: Rdclark
Re: MultiFinder bug???

Spud,

Basically, background applications shouldn't be putting up alerts — that is the privilege of the foreground application. You should use the Notification Manager instead, (which can put up an alert in front of the frontmost application if you really have to.)

The Dialog Manager assumes (correctly) that modal dialogs (and alerts) will always be the frontmost window, and so have no redrawing code.

...Richard

From: Rguerra
Re: Dialog Redrawing

Ah, but that doesn't mean that modal dialogs shouldn't have redrawing code! What about the case when a modal dialog is in front and a screensaver kicks in? If you don't handle the update event, the user is left with a blank dialog except for whatever buttons, etc. that were part of the DITL item list. If you need to draw frames, outline buttons, etc. you can either add a `UserItem` to the DITL (a custom drawing routine) or handle updates in a dialog filter proc that you pass to `ModalDialog`. I agree about using the Notification Manager from the background. You might want to make sure your program is MultiFinder-Aware and keep track of whether you're in the foreground or background and call your normal dialog in the foreground and the Notification Manager dialog in the background or else do nothing until the user brings you to the foreground again.

Rich

From: Rdclark
Re: Dialog Redrawing

Ah, but that doesn't mean that modal dialogs shouldn't have redrawing code! I agree — but who's going to write the ROM patches to do it <grin>? Really, you have to assume that a modal dialog will be responded to immediately, and that your screen saver probably won't have the chance to kick in. (By the way, there is `_limited_` redrawing code for alerts, since your button's contents and any static text will be redrawn...just not the icon or the outline around the button.)

...Richard

From: Mark Worthington, Worcester, MA
Subject: XCMD Intercepting mouseUp

I have written an XCMD which is called within a `mouseDown` handler in a HyperCard script, which tracks the mouse until it is released. No problem. Recently I decided to allow the user of the XCMD to be able to specify a parameter by which he could choose to have the XCMD intercept the inevitable subsequent `mouseUp` event, or let nature take its course and have HyperCard receive the event and do with it whatever it was supposed to. I therefore put a `GetNextEvent` loop at the end of my XCMD, expecting this would intercept one `mouseUp` event and then exit the XCMD. But it doesn't intercept the `mouseUp` at all. I have added several diagnostic lines of code and found that the loop really is not finding a `mouseup`. I gave the loop 10 seconds

to receive such an event. I also put a `mouseUp` handler in the button to beep 5 times. Lo and behold, when the 10 seconds had elapsed, the button beeped 5 times, and my diagnostics reported no `mouseUp` was detected by the XCMD. Now I know HyperCard 1.2.2 is supposed to change HyperCard's habit of sending itself a `mouseUp` message *before* removing the `mouseUp` event from the queue (oh, yes. I am using HC 1.2.1), but that doesn't seem to be the problem. What is HyperCard doing to me, and how? Would using `WaitNextEvent` make any difference, and if so, why? Please help. Thanks so much.

From: thecloud (Ken Mcleod, La Habra, CA)
Subject: trouble Intercepting mouseUp

You might try setting the system event mask to enable `mouseUp` events yourself (using `SetEventMask()`...see the Toolbox Event & OS Event chapters of IM), since it sounds like HyperCard is disabling `mouseUp` events before calling your XCMD. Alternatively, you might want to try calling `GetOSEvent()` instead of `GetNextEvent()`, since the system can intercept events gotten with `GetNextEvent()` before your code ever sees them.

Somebody more familiar with HyperCard should be able to give you a better answer (or else an 'official' one); in the meantime, hope this proves helpful.

From: macww (Mark Worthington,)
Subject: trouble Intercepting mouseUp

Ken, Thanks for the reply. I'll try your ideas. In the meantime, if anybody else has more specific knowledge of what HyperCard is doing, or where else I might find out, I would appreciate it. Even if one of Ken's suggestions works, it is always unsatisfying to "fix" something without understanding what got "fixed." Thanks again, Ken.

Volume 5, Number 7

From: Rguerra
Re: cdev "shell"

It occurs to me that writing and debugging `cdev`'s would be greatly simplified if the programmer could simply place the `cdev` unit into a supporting shell program that would emulate the function of the control panel. This shell would send the appropriate `cdev` messages to the user's `cdev` unit. The programmer using LSC, LSP, or even SADE could step through the instructions of the `cdev` as per usual with an application. Has anyone ever seen such a beast? If so, would you be willing to share it with the rest of us? If one doesn't exist yet, what would go into writing one? Any comments? Rich

From: Ctlil
Re: Res Error Proc (Fw by Sysop)

Anybody know how to redirect `ResErrProc` to a light speed pascal routine? The following code does not work.

```
ResErrProc = ^ longint;  
ResErrProc = pointer($0aF2) ResErrProc = 0rd4(0MyErrorProc);  
Procedure MyErrorProc ;
```

```
begin
  Code := 3 ;
end; (by the way Code is a global variable)
```

From: Poltik
Re: AppParmHandle

Have you ever played with the Finder Info Record, a handle to which is found in AppParmHandle(\$AEC)? I'm trying to change the record between application launches. Here's what I'm doing:

```
AppParmHandle = $AEC;
FInfoRec = record
  mymez: Integer;
  mycount: Integer;
  volref: Integer;
  mytype: LongInt;
  versnum: Byte;
  notused: Byte;
  namelen: Byte;
  myname: string[30];
end;
FInfoRecPtr = ^FInfoRec;
FInfoRecHnd1 = ^FInfoRecPtr;
myParms := Handle(AppParmHandle);
( $AEC - low mem glob )
HLock(Handle(myParms));
myF := FInfoRecHnd1(myParms);
myF^.mymez := 0;
myF^.mycount := 1;
myF^.myname := 'Jean';
etc....
HUnlock(Handle(myParms));
```

I can see everything changing in Lightsbug, but for some reason it crashes after the volref is changed Any ideas? Help appreciated...

From: CtlI
Re: ResErrProc

Redirection of the ResErrProc pointer needs to be a longint instead of just an integer. Also, in Lightspeed Pascal, any file function such as Reset, Rewrite or write doesn't seem to behave nicely. If the disk is locked, there is no invalid return, rather you are dumped out to the system with a id=08.

From: Nicks
Re: My Cdev Is a little confused.

I'm writing a cdev in LightSpeed C, and having a weird problem. If I try to change the state of the Pen with a PenPat or PenNormal call, the pen pattern gets set to garbage. A short sample might be:

```
PenPat(grey);
FrameRect(&anyRect);
```

- This will fill the specified rectangle w/ garbage. I assume that the problem has to do w/ the fact that I'm running this code inside a DA, and I can't get to the QuickDraw globals? Or am I on the wrong track completely? Thanks in advance.

From: Batlsta
Re: My Cdev Is a little confused.

The problem is that you are trying to use a QuickDraw global

when in fact you don't have it. When Think C runs it allocates these globals for QuickDraw and you initialize them when you do the InitGraf(&thePort); in a cdev there is no such a thing, therefore when you do the PenPat(&gray) you are dereferencing who knows what. What you have to do is make up your own pattern.

```
Pattern myGray;
myGray[0] = myGray[2] = myGray[4] = myGray[6] = 170;
myGray[1] = myGray[3] = myGray[5] = myGray[7] = 85;
```

Then do a PenPat(myGray); and you'll get the result you want. Ricardo

From: Nicks
Re: My Cdev Is a little confused.

Thanks for the info. I assumed that it had to be something like that. Questions: Does this affect other things besides PenPat? Is there a way to "find" the QD Globals that the Control Panel is using and address those? Where in IM is all this addressed? Thanks again.

From: Rdclark
Re: My Cdev Is a little confused.

A better way to fix the "bad patterns" problem is to set up A5 at the start of your code and restore it when you're done. If you do this, A5 will point to the start of the current Application's globals, and the end of the current QD globals — which is exactly where it should be when you need to use QD. ...Richard

From: Nicks
Re: My Cdev Is a little confused.

Yeah, that's what I was assuming. BUT: I'm a bit new a this "where's my globals" stuff. How do I get the right value of a5? Get it from CurrentA5, then use in-line assembly (which I have no clue how to write)? Or can I use SetupA5 & RestoreA5? IM says these are for interrupt handlers only. Do they have other uses? BTW: my work-around for the original problem was to just use a PICT resource with a grey pattern in it, the use CopyBits to blast that into whatever rectangle or whatever I needed to use it in. Ugly, huh?

From: Nolsy
Re: MacPaint II diagnostics

Having recently attacked MacPaint II with MacNosy (the ONLY programmer's tool), I kept encountering DIAGS everywhere. A rainy Monday evening with the wife at work gave me the time to track it down: it seems that Claris neglected to remove about 4K of diagnostic code, basically because it is thoroughly embedded everywhere. It doesn't seem to do much, apart from double-check various drawing actions, but it *can* be enabled by patching the application: Using any decent File-Mangler (I used FEdit+) do the following: Replace: 6CE6422D E59349ED with 6CE61B40 E59349ED There should be only one occurrence, and never patch your original, blah, blah. Cheers! Noisy

From: Rguerra
Re: MacTutor Index (Fw by Sysop)

Is there an index for the fourth volume of the collected articles of MacTutor. I've got all the original issues but I'd like an index. How about posting one here, or even publishing an

index to a volume in the first issue of the succeeding volume?

[MacTutor now sells an index a text with tab formatting for adding into your favorite database application. It is a nominal fee (\$2 plus s/h) especially since all the work I've put in to prepare it initially.-KC]

From: Dale

Re: A sick MPW text file??

I'm not sure what category this falls into, but here goes: I've just run into the strangest problem with MPW 3.0. It seems that editing the file SANEMacs881.a in the MPW Shell and then trying to Revert the file explicitly causes the Shell to hang. This happens with any copy of the file that's simply a Finder copy or an MPW Duplicate of the original, but there's a bizarre way to create a healthy copy: first create an exact copy of the SANE-Macs file, then open it and Catenate the text of the original file into the open window. If you save that the Shell will behave normally when you edit it. Interestingly enough, if you simply create the copy with Catenate instead of first copying the entire file, Compare reports that the two files don't match anywhere! Very strange, since (as far as I know) Catenate simply transfers the data fork and Compare just compares the data forks. What's even stranger, using Catenate without opening the file first doesn't work: your copy will hang the Shell as before! Do any of the experts out there have a clue as to what's going on here?

From: Macrobln

Re: MacTutor Listings Font

This is a pretty simple question. What is the Font used for Program Listings in MacTutor. I have been looking for a nice clean sans serif, monospaced Font for a while. MacTutor program Listing appears to be just what I want. What is it and where can I get it. Thanks.

[MacTutor uses a special LaserWriter version of Monaco call Thin Saratoga that was created especially for program listings by Michael Mace, now at Apple Computer. We might let you have it if you promise not to distribute it to our competition! -Ed]

Volume 5, Number 8

From: Caro

Subject: mac logic board

I have a problem with my Mac+ logic board (I already KNOW it's not RAM or any peripheral)

- on power-up it puts up a black screen and sad Mac
- doesn't even get as far as the RAM test
- and posts an error code of 030020.

Apple doesn't support chip level repair, and I really don't want to pay 350 plus trade in for what is probably a 2.00 chip. Any Mac hardware hackers out there? I'm a programmer and my boss is equally expert with hardware and software. Anyone know what this error message means? Other handy hints? Where to get a complete schematic?

From: Smugwimp

Subject: mac logic board

The chances are it is one of your ROM chips. You should have snap in chips as far as the ROM goes. Call "Pre-Owned Electronics" (sorry I don't have #). They have ads in the "Computer Shopper" and sell them for around \$125 a set. If you do change them yourself

a) Be certain you are out of warranty

b) is Applecare not an answer ?

c) if you DO IT, be SURE to observe anti-Static procedures so as not to blow the rest of your Mac replacing one section. —
> SmugWimp

From: Shunt

**Subject: Xcmds and code greater than 32K
HELP!!**

I'm in the process of writing a series of XCMDs for HyperCard; having got right through to the 49th - now I've have run into problems.

The code for this particular command is too large for a standard 32767 byte segment and, on linking, I have a stream of error 48 messages whereby the PC-relative edit offset is out of range (jump table calls greater than 32k away). I have tried reorganizing the procedures, functions and include files - unfortunately I cannot strip out any more of the code. I can set the segment size to accommodate the code OK, and have tried -srt option on linking and -b option on compiling: All to no avail.

Any suggestions please! These Xcmds will also be duplicated as 4D External so any Help on this front would also be greatly appreciated.

Many thanks Doug Hunter.

From: Craigk

Subject: Xcmds and code greater than 32K

I had a similar problem programming the Manhattan ZIP Code Locator. In trying to handle over 800 streets and thousands of addresses, I just resorted to 3 XCMDs. If what I wanted didn't get done in the first one, I just called the next one. I realize that this is not a direct answer to your question, but that is how I worked around the 32K limit.

From: Spud

Subject: XCMDs with CODE?

I was about to write an XCMD that calls an external CODE resource, when someone informed me that HyperCard can't do that!!! Is it true???

From: Rguerra

Subject: KeyCode Question

I need to translate a keyCode (obtained from the EventRecord message field using the KeyCodemask and BitAnd) BACK into an ASCII character. I can't just get it from the Event message because modifier keys will give you strange characters and I just want exactly what's on the key that was pressed. I tried using KeyTrans but I can't seem to make it work. Can anyone help me with this!?!? I'm up against a deadline and I need to get this problem solved. Thanks!

Rich

From: Wenbrow

Subject: List Manager Follies

Does anyone out there use List Manager?? I have been struggling to write a simple two-dimensional list for data display and entry. I have succeeded in using Lnew to create a list and attach it to a window. I have used Lsize to adjust the size of the list when the window is resized, and several other standard tricks. Here's the problem: When the mouse is clicked inside the content region, my program crashes with an "Address error." The offending statement is:

```
TempBoolean:= Lclick (NewPoint, event.modifiers, ListHand)
```

Amazingly, this statement works fine when the mouse is clicked inside the scroll bar! The list scrolls beautifully. But when it's clicked inside the displayed list itself, the Address Error occurs. I have checked the values of NewPoint.v, NewPoint.h, event.modifiers and ListHand (the handle to my list), and all values are reasonable! My program is translated from a Lightspeed C program which appeared in MacTutor in March 1989: "How to Write a Spreadsheet in LS C."

From: Skaros

Subject: VBL Inits

I've been trying to install a VBL at init time but have not been successful so far. My strategy has been to compile a simple VBL(something that posts an APP1 event) as a 'CODE' resource. Then my installer sets up a test window and installs the VBL in the following way: 1) get the resource, 2) lock the handle, 3) get its size, 4) allocate a block of memory with a 'newptr' call, 5) do a 'blockmove' of the dereferenced locked handle of the 'CODE' resource into the newly allocated block of memory. Of course, newptr gives me a block in the application heap, but this is just a pilot program. I could easily define a 'newsysptr' call with inline code to get me a block in the system heap. Anyway, my VBL installs, interrupts once and then nothing. When I quit my installer, I get a 'qerr' this means my VBL isn't in the queue. My guess as to what is happening is that my vbl counter that is supposed to be initialized during every cycle of the VBL task with the statement—vbl_rec.vblCount :=10; really isn't working. After all, how can a VBL task that is moved into a heap with 'blockmove' ever know where its VBL record is. Therefore, it doesn't reset its counter and therefore it expires after one interrupt. How do I pass the address of the VBL record used in the installer program to the VBL task so that after the VBL is installed, it resets its counter? Or, is there another way to set up a VBL task as an init?

Stefan Karos

From: Rdclark

Subject: Window Mgt, THINK Pascal 2.01?

There are several common problems you can have when resizing a window. The First, the 4 lines you'll need should look like this:

```
SetRect(sizeLimits, 32, 32, screenBits.bound.right,
screenBits.bound.bottom);
result := GrowWindow(whichWindow, theEvent.where, sizeLim-
its);
```

```
if (result <> 0) then
  SizeWindow(whichWindow, LoWord(result), HiWord(result),
TRUE);
```

Now, for the problem areas: 1) your "sizeLimits" rect is undefined, or has a minimum size (the first 2 numbers) of (0,0). 2) TheEvent.where is given to you in GLOBAL coordinates, which GrowWindow also expects. Are you calling "Global-ToLocal", perchance? (If so — don't!) 3) LoWord() and HiWord() could be reversed. 4) You're not checking to see if the result is non-zero (0 = "no change!") before calling SizeWindow. 5) The window's defProc is set to be a NoGrowDocProc (which sounds like something you'd order at a Chinese restaurant) instead of the more common "DocumentProc".

Enough of a hint?

...Richard Clark

Apple Developer University

From: Wenbrow

Subject: List Manager Scrolling

I am looking for a "hook" that will enable my program to respond to scrolling of a two-dimensional List, so I can rewrite some column and row headings on the screen *as the list is scrolling*. List Manager's LCLICK function has a hook, stored in the LClickLoop field of the list, that the programmer can point to a custom routine. Inside Macintosh IV says that the routine will be called while the mouse is clicked or held down in the list rectangle or the scroll bars, but it does NOT work properly in the scroll bars. The programmer's custom routine only gets called when the mouse is clicked in the list's visible rectangle, not the scroll bars.

This problem is discussed in the article "How to Write a Spreadsheet" in the March 1989 issue of Mac Tutor. The author says this problem can be fixed by installing a pointer (to your custom routine) in the system global variable DragHook (\$9F6). Well, I finally succeeded in installing the pointer, but it only works in the wrong situations: My routine is called while the scroll bar's "thumb" is being clicked or dragged, but it is *not* called while the mouse is held down in the scroll bar's "arrows." Thus, I have the same old problem: I can't update my row and column headings until the mouse is released. This looks tacky and confusing to the user. Is there some other approach to solve this problem, or do I have to give up on List Manager and display all my data using QuickDraw?

From: Ivano

Subject: Help Compiler using Pascal

I am trying to use Robert Monsen's Help Compiler in a MPW Pascal program. The calling procedure written in TCU is as follows:

```
ShowHelp()
(
  Handle rsrcH;
  ProcPtr pp;
  if (rsrcH = GetResource(CODE, 2000))
  (
    HLock(rsrcH);
    pp = (ProcPtr) *rsrcH;
    (*pp)();
    ReleaseResource(rsrcH);
```

```

}
}

```

In rewriting this to pascal, I am having trouble with the (*pp)(). If my limited C experience is correct, what I need to do is go to the address of CODE 2000 pointed to by pp. Pascal isn't to friendly with ProcPtr's, any suggestions would be appreciated.

Ivan L. Ossander

From: Gurglekak

Subject: Help Compiler using Pascal

Here's what I use to jump to the external transfer protocol code resources in my terminal program:

```

procedure LaunchProto (transType: resType);
var
  protoName: Str255;
  theProto: Handle;
  theProtoID: integer;

procedure CallProto (serialInput, serialOutput: integer;
  itsID: integer; macBinary: boolean; p: Ptr);
  inline
    $205F, $4E90;

begin
  GetItem(GetMHandle(theMenu), theItem, protoName);
  theProto := GetNamedResource(transType, protoName);
  if theProto = nil then
    Message(noProtocol, cautionIcon)
  else begin
    GetResInfo(theProto, theProtoID, transType, protoName);
    HLock(theProto);
    CallProto(serialInput, serialOutput, theProtoID, true,
theProto~);
    HUnlock(theProto);
    HPurge(theProto);
  end;
end;
end;

```

I'd show you the original note from THINK I got on CompuServe, but CIS might take offense.

From: Bryce

Subject: Setting current folder

I am using a filter from SFPPutFile to allow the user to create a new folder. Does anyone know how to enter the created folder from within the filter? The code I'm using is:

```

function MyPutFileDlg (item: INTEGER; theDialog:
DialogPtr): INTEGER;
( This filter is used to allow the user to create a new folder
from within SFPPutFile )
const
  CurDirStore = $398;
var
  topLeft: Point;
  curDirPtr: ^Longint;
  dirParmBlk: HParmBlkPtr;
begin
  case item of
    -IntOne, getOpen: begin
      firstCall := true;
      MyPutFileDlg := item;
    end;
    newFoldButton: begin ( RNew FolderS button pressed )
      topLeft.h := 90; ( top left horiz coord for
Put File dialog )
      topLeft.v := 75; ( top left vert " " "

```

```

" << hit any key >>" )
  firstCall := true;
  SFPPutFile(topLeft, 'Enter New Folder
name:', 'Empty Folder', NIL, Reply, NewFoldID, @MyFilter);
  firstCall := true;
  if Reply.good then begin ( Create the design-
nated folder )
    curDirPtr := Pointer(CurDirStore);
    putFileVRefNum := Reply.vRefNum;
    with dirParmBlk^ do begin
      ioCompletion := NIL;
      ioNamePtr := @Reply.fName;
      ioVRefNum := putFileVRefNum;
      ioDirID := curDirPtr^;
    end;
    theErr := PBDirCreate(dirParmBlk, false);
    MyPutFileDlg := 101; ( Tell SFPPutFile to
update file list )
  end;
  otherwise
    MyPutFileDlg := item; ( Pass control back to
SFPPutFile )
  end; ( case )
  I've seen this done in Raymond Lau's SFVol init. Bryce

```

From: Nicks

Subject: WMgrPort ?'s

How do DA's such as BackDrop update the whole screen? I wanted to try the Disk icon INIT from the May MacTutor in real time, changing the icon, then informing the Mac that the screen was invalid. I tried getting the WMgrPort (yeah, I know this is a no-no, but I'm just spelunking for fun, not publishing), setting the port to what was returned, getting the portRect, then calling InvalRect with it. Boom! Same thing happens when I get the GrayRgn and call InvalRgn.

Sample:

```

GrafPtr oldport, wport;
Rect r;
GetPort(&oldport);
GetWMgrPort(&wport);
SetPort(wport);
r = wport->portRect;
InvalRect(&r);
SetPort(oldport)

```

The code appears to blow up in the InvalRect call in the ROM in EqualRgn with a odd address. Am I missing something stupid, or is there another approach to doing this? Thanks in advance.

From: Batista

Subject: My Cdev is a little confused.

The Control panel may not be using globals, but, if it is, then it is using them of A4. That of course is of no help and you should not even try to use anything that belongs to the control panel. If they happen to have some globals addressed of A4 you don't even know what they are. The problem you had with the pattern would apply to anything that is one of the "QuickDraw Globals" in an application such as color and patterns. The easiest way to get around this things is to make your own variables when you need them. The AppleTalk project of LSC makes a socket listener to

get requests, and it is inside the code, so don't try doing things with AppleTalk in resource that may get moved like cdev's.

Ricardo Batista

From: Macww

Subject: My Cdev is a little confused.

Make sure you take a look at both Tech Notes 180 and 208 if you hope to do SetupA5() and RestoreA5() in a MultiFinder environment. I think they both used to be on Mousehole Download. I am just now trying to get used to this new BBS host software, so I haven't check the files section to see if those tech notes are posted. If not, leave a note requesting them.

From: Pkram

Subject: file busy

I have adapted the text editor in "Using the Macintosh Toolbox with C" for Aztec C 3.6b. However whenever I save a document created by this editor, other applications such as Word 3.01 or Edit, cannot open the document without getting an alert stating that the file is busy and can only be opened with read only permission. I then used MacTools and checked the busy flag. The file according to the busy flag, was NOT busy. Where in IM can I find the answer to this problem?

From: Fischer

Subject: Editing in a scroll back buffer

I am currently developing a DA with X/Y Modem file transfer, Hayes compatible modem control, and Scroll Back Buffer, in MPW C. I have reached the point where I have to implement a text selection (Copy, Cut, Paste ...) in my Scroll Back Buffer. I haven't found any concrete idea on how to do this in MacTutor. I feel frustrated because I have to 'reinventer la roue' for this. Is there anybody here who could help me ? Thank you. Pierre.

From: Mike

Subject: C-Ing is not believing!

I have recompiled some XFCN's written in LightSpeed C 2.0 using LSC 3.0. They don't work! Although they compile OK, the call backs to Hypercard don't seem to work all the time. For example,

```
paramPtr->returnValue = (Handle) CopyStrToHand("A string");
always gives an empty hypercard XFCN result. Anyone
have any ideas about what on earth I am doing wrong?
```

From: Siegel

Subject: C-Ing is not believing!

In TLSC 3.0 and later, strings are treated as global variables, rather than being placed pc-relative. Therefore, you need to do a RememberA0() and a SetupA4() at the beginning of your main, and a RestoreA4() at the end. The same applies for floating-point constants.

From: Apage

Subject: Smalltalk/V Primitives

I am about to embark on some development with MPW

assembler to generate a new set of floating point primitives for Smalltalk/V Mac. By using TMON to set a trap intercept for _FP68K I found that Smalltalk is still trapping through SANE to get to the 881 processor. I am interested in seeing if I can generate faster primitives in Assembly that will address the coprocessor directly. I am doing this both to see if there is benefit to be gained and to get the feel of the interface in Smalltalk and its primitives. Would anyone be interested in my results and experiences?

From: Nolsy

Subject: SysEnvirons help?

OK: so not all of us rush out and buy MPW 3.0 as soon as it's released. Can anyone help out a (starving) LSC coder by revealing the SysEnvirons codes for the MacSE/30 & MacIIcx? Pretty please?

From: Batista

Subject: My DA drawing don't work! (sob)

Check the messages on C, there was a similar case. The work around is to declare your own gray pattern.

From: Sped

Subject: Dialog Redrawing

In every serious application I've ever written, I've run into this problem. Inevitable, I write a "FlushUpdates" procedure that goes thru each of my windows and redraws those that have non-empty updateRgn's. For dialogs (modal and otherwise), a call to DrawDialog does the trick. Clears up all kinds of ugliness, such as when one modal dialog appears in front of another.

From: Skarosx

Subject: lhe

This is a neat board. A lot of programming info in one place is a time saver. I downloaded some of the code snippets and have been trying to install a VBL task as init. Well, I ran into a small problem. To get a VBL to live in the system heap, it seems I have to get a pointer to a block in the Sysheap. Then I can blockmove the VBL code to that pointer. After speaking with the helpful folks at Symantec, the call one needs to make in pascal has to be defined in assembly or with inline code. No problem, it is:

```
Function newPtrSys (N: longint): ptr;
Inline
```

```
$201F, $A51E, $2E88;
```

Mucking around with the system heap got me nowhere, so, I tried to get the technique to work with the application heap. First, an innocuous VBL task (something that just posts an APP1 event) is compiled separately as a code resource with a specific ID. Second, an application is written to install this VBL code in the application heap, and then the application checks for APP1 events in its event loop. The commands to set up the VBL are:

```
resH := GetResource('CODE', 600); (This TcodeT resource)
(with ID = 600 contains the compiled VBL task )
```

```
Hlock(resH); (Lock the handle)
```

```
resSize := GetHandleSize(resH);
```

```
Aptr := newPtr(resSize); (Get a pointer on the appHeap to
a block big enough to accomodate theTcodeUresource)
```

```
Blockmove(resH, Aptr, resSize); (Move the TcodeU tothere)
The rest of the application finishes setting up a simple menu
```


and window and the VBL is set up with:

```
VBL_ptr := @vbl_rec;  
With vbl_rec Do  
  Begin  
    qType := ord(vType);  
    vblAddr := resH; (the VBL task is a code resource  
    (moved into the address at sysptr)  
    vblCount := 60;  
    vblPhase := 1;  
  End;
```

```
os_result := Vinstall(VBL_ptr);
```

With this program, all I get is one interrupt. Quitting the app gives me 'qerr', i.e., the task is not installed! Help! Stefan Karos

From: Nicks

Subject: VBL

Sounds like you're not setting the vblCount back to a non-zero value inside your VBL task. A quote from IM: "VBLCount specifies the number of ticks between calls to the task. This value is decremented each [tick]" until it is zero, then the VBL task will be called "The task must reset VBLCount, or its entry will be removed from the queue after it has been executed."

From: Skaros

Subject: VBL

Right! My guess as to what is happening is that my vbl counter that is supposed to be initialized during every cycle of the VBL task with the statement—vbl_rec.vblCount := 10; really isn't working. After all, how can a VBL task that is moved into a heap with 'blockmove' ever know where its VBL record is. Therefore, it doesn't reset its counter and therefore it expires after one interrupt. How do I pass the address of the VBL record used in the installer program to the VBL task so that after the VBL is installed, it resets its counter? Or, is there another way to set up a VBL task as an init?

Stefan Karos

From: Rdclark

Subject: Scrolling

Using SetOrigin() is probably the _worst_ possible way to do scrolling, as it causes your scroll-bars to flicker and generally behaves in strange ways. If you are using object-oriented programming, as you imply, it should be easy to base your display objects on a "display device" type which contains the (virtual) top-left of the GrafPort and redraw all objects in relationship to that. Still, if you still want to use the SetOrigin method, drop me a line at rdclark@applelink.apple.com (or here as "rdclark"), and I'll dig up some code.

From: Gurglekat

Subject: Souping up TE

Don Brown managed to soup up TextEdit for his terminal DA MockTerminal. I've put trap breaks on all the relevant TE traps, and he seems to be using TEInsert for everything but backspaces and TEKey for backspaces. I don't know why he made the distinction, because there doesn't seem to be any speed benefits. Maybe there are on the 8 MHz Macs. But in any case,

when I use the same strategy in my code, the screen i/o can barely keep up with 300 baud, if that. What's DB's secret? I sure don't want to write a TE clone.

From: Frankh

Subject: Souping up TE

If you want to use TextEdit to display data any faster than 300 baud, you'll have to buffer the incoming data, giving time for TE to digest it. The easiest way to do this is to increase the serial port input buffer size to something reasonable - the default 64 byte buffer just isn't enough. Use SerSetBuf to increase this buffer to 512 or 1K bytes. Don Brown is probably using TEKey to backspace over text that already has been displayed and is no longer in his buffer. It's the easiest way.

From: Gurglekat

Subject: Souping up TE

Hmmm. Well, what I meant about the distinction was that I wondered why he used anything but TEKey. Whatever. Now, what's this about buffering data? I've got a 10K serial buffer. Should I only get characters from it when it's full to a certain level and then get a lot of them? At the moment I'm getting a character each time through the main event loop. And emptying the buffer entirely each time through the main event loops doesn't seem to help either.

From: Frankh

Subject: Souping up TE

If you use the SerSetBuf function to increase the serial port buf, then all you have to do is do a SerGetBuf (which returns the # of bytes in the input buffer) once through the main event loop. If the input buffer (the one increased with SerSetBuf) has data in it, just read off however many bytes were received (usually into another buffer) and process them at your leisure. Since you may end up with more than one character, it's easier to display this data by using TEInsert. Usually, you'll handle backspace and other control characters before you sent the buffer out to TEInsert, but if you only have a BS in the buffer, then you have to use TEKey to make TextEdit backspace for you. TEInsert ignores BS.

From: Gurglekat

Subject: Souping up TE

Not only does it WORK, but it works in a neat way. As soon as I added the scrollbar for the scrollbar buffer, I had to update the scrollbar control bar max and the value to match every time a new line came in. So now, instead of the output jerking along like some sad approximation of multitasking, most of the buffering goes on during the new line sequence, so lines appear all at once but partial lines are also taken care of! Thanks again.

From: Inbox

Subject: Scrolling

I'm also very much interested in getting some good scrolling code. I'm writing a spreadsheet-type of thing, and I can't redraw the whole screen every time - takes too long. So, please e-mail any scrolling code at all. Thanks in advance,

Volume 5, Number 9

From: Cheasy
Re: INIT31 Help wanted

Today I'm writing because of some VERY difficult programming problems. At this time, I am trying to write an INIT for use with the INIT 31 mechanism, which will do something like pop up a dialog box onto the screen. (Yes, indeed, I need this during startup!)

Unfortunately, if I call InitWindows from the toolbox, the startup screen is destroyed and the menu bar will be drawn, along with a fresh erased desktop. This is an very ugly effect, especially if there are more than one or two INITs in my System Folder. So I wrote some code to draw some pseudo-dialog boxes, fill them in, handle something like EditText fields and look for mouseDown events in different regions. They all work nice in my testing program. To make them work, I have to initialize a lot of low-memory globals myself, including my own quickdraw globals. This all worked fine.

But when I try to CopyBits the background of my window before drawing onto it, the routine will crash immediately. Nothing works, and even my own QD globals seem to be destroyed. Has anyone out there a little bit experience dealing with quickdraw at startup (INIT31) time? I am interested in any code which shows how to set up and reference to Quickdraw globals at INIT time.

P.S. I apologize my bad english, but in fact, I am only speaking Pascal. (You can ask me for a German version of this bulletin.)

From: Anthony
Re: INIT resources

I have a question, I am working with INIT's, and just wanted to see if this bit of code would really do the job. It is written in LS C, and is all of one line,

```
main()
{
    Sysbeep(5);
}
```

I have saved it as an INIT, but when I add it to the list of INIT resources in the system, and reboot, it doesn't do anything? Why, I don't want to use the INIT 31 trick, I want this to be a part of the system file, so I don't have to worry about users carrying it over with them when they switch pc's.

From: Retzes
Re: INIT resources

I think there are two possible reasons why the INIT didn't work. One, at the time the init is called the Sound-manager, Operating-System Utilities, or Sound-Driver has not been initialized. Two, the INIT didn't get called for some other reason. When making an INIT that uses Quickdraw functions, I think you have to call InitGraf with a pointer to some local variables that resemble the Quickdraw globals. This was what seemed to be

necessary when I added a function that displayed the ICN# for the INIT at the bottom of the screen at startup. I found out how to do this with TMON, examining another INIT file. First you must decide what you want your INIT to really do, then decide if its possible with the system existing at startup.

From: Mikec
Re: New folders from HyperCard

Is there any way to create new folders form within HyperCard using an XCMD? I have a HyperCard application that needs to create a new folder with the user's name every time a new user registers. There is nothing in "Inside Mac." or the Mac Tech Notes on this. Is this a closely guarded secret of the Mac's finder? Thanks for any info or support.

From: Tonys
Re: INIT using dialog manager

How do I create an INIT using LS Pascal which accesses the dialog manager, such as a simple StopAlert call? I think I know how to access the Quickdraw globals using CurrentA5, but once the dialog has been dismissed, the system bombs. Am I missing some other initializations or have resources labeled incorrectly?

From: Siegel
Re: INIT using dialog manager

You'll need to call InitWindows, InitFonts, and InitDialogs as well. It's generally un-INIT-like to fire up the Window Manager. If you want to give a message, use ShowInit to draw your INIT's icon with an X in it, or something similar.

From: Rguerra
Re: KeyMap Kraziness

Can someone PLEASE help me with manipulating the keyMap and modifier keys in assembly? I have user-definable modifiers stored (derived directly from the event record's modifiers field. I know that the modifier keys all live in bytes 6 and 7 of the keymap. I can easily MOVE these bytes to a data register, but I can't seem to figure out how to manipulate the bits of either the stored modifiers or the keyMap bytes, and then determine if they are the same. I have to do this directly in assembly since this is in a patch. I know that this will only take about 4 lines of code, but I can't quite get it right. (It's so frustrating since I can do the check so easily in Pascal.) Can someone help? I'd appreciate it.

From: Gurglekat
Re: KeyMap Kraziness

Give the BTST instruction a whirl. I'd tell you more about it but my Motorola 68000 manual seems to have escaped me.

From: Siegel
Re: More Help

To create your own port and fill up black, try this:

```
main()
{
    GrafPort myPort; /* static storage */
    InitGraf(&thePort);
    OpenPort(&myPort);
}
```

```
PaintRect(&myPort.portRect);
}
```

From: Apage
Re: More Help

Your most obvious problem in this code is your calls to `EraseRect` and `FrameRect`.

The Pascal traps are expecting them to be passed as a Pointer to rectangles, however the C compiler will try passing all 16 bytes of each rectangle onto the stack, as per C calling parameters. Change your calls to the following and try again:

```
EraseRect(&myPort.portRect);
FrameRect(&myPort.portRect);
```

From: Bcbg
Re: More Help

When you call the ROM in C, you must be aware of the standard calling conventions. Every object more than 4 bytes in size or every VAR parameter is passed by "address" and not by "value". It means that to call `EraseRect`, you must do:

```
EraseRect (&myPort.portRect);
Same thing for FrameRect. If you used LSC 3.0 or MPW C
3.0, the compiler would tell you these problems...
```

From: Cforden
Re: LSC Assembly: Save registers for call?

What registers must I save before I call another routine? Inside Mac (phone book ed.) and LSC (3.0 manual) both hint that registers D0-D2 and A0-A1 are generally considered expendable and are generally trashed by routines. However saying that they (LSC and the toolbox) trash them is not quite the same as saying I am allowed to trash them or let them get trashed. In fact I found that if I did not save those registers prior to calling my LSC `wordBreak()` routine and restore them upon return, the toolbox would get confused and crash the system. I couldn't find explicit guidance in Williams' book "Programming the Macintosh in Assembly Language" (Sybex), but many examples showed all registers being saved by routines.

From: Nolsy
Re: LSC Assembly: Save registers for call?

Hmmmm...I thought that the LSC (v3.x) manual (pg. 151) was pretty clear about saving D0-D2/A0-A1, and IMvI says the same. As for what an application can trash, I haven't had any problems trashing registers prior to Toolbox calls. Like yourself, though, I did get into trouble with 'call-back' routines: routines that are provided by the application, but actually called by the Toolbox. (The one that bit me was a List `clikLoop`) I have adopted the 'better safe than sorry' attitude of pushing/popping ALL registers in call-back routines, just to ensure future compatibility if nothing else. The space and time wasted by pushing registers is nothing compared to the agony of trying to figure out which one got trashed by whom!

From: Siegel
Re: LSC Assembly: Save registers for call?

The conservative way to go is to save any registers you use, and if any routines that you call use them, be sure they're correct.

From: Bruceq
Re: Alternate Screen/Snd

I am currently writing a game for the Mac and need to use the Alternate Screen/sound buffers to make it playable on non 020 machines. I know that I can use the alt. buffers if I force the user to boot off of the master disk, but I would really like to have the game load off of the users harddisk if possible. The problem is the case of MultiFinder running. Does `_Launch` still allocate the alt buffers for you when asked? What if another program had loaded previously? Will MultiFinder reserve that space or swap it out if the foreground application needs the alternate buffers? Or has Apple decided not to support the alt buffers in the future. I don't need to use a frame buffer on a 68020 or 030 machine because of the extra HorsePower, but doing full screen animated graphics on a Plus or SE with out the frame buffer is the pits.

Any ideas or higher knowledge?

From: Gurglekat
Re: WMgrPort ?'s

There is a low memory global called `DeskHook`, \$A6C, which according to my Inside Mac X-ref is "Address of procedure for painting desktop or responding to clicks on desktop." You might ask jbx, whose CompuServe address is 73177,1404, how he did the pop-up menu bar in `hierDA` — I'm pretty sure he used this routine. If not, you might want to contact the `DeskPict` people.

From: Gus
Re: ANIMATION

Hey there guys. I'm writing a freeware game for the MAC and am interested in how you guys do flicker free animation?? I've tried everything. Is there some secret I don't know about?? Code would really be appreciated. Also I'm using a Mac+ and I'm, writing in LSC. I would really appreciate any help as you guys are my last hope before that programming black hole.

From: Inbox
Re: ANIMATION

You can read about it in Scott Knaster's 'How to Program Macintosh Software'. Anyway, here's the advice given there: (I'll condense it...) It says that the first thing that occurs in the vertical retrace interrupt, is that the global variable `Ticks` gets incremented. So you can draw right after the interrupt occurs, before the beam actually draws the stuff on the screen. So, here's the suggested code of your drawing proc:

```
Var tickValue : longint;
begin
  tickvalue := TickCount; (calls ROM to get value of Ticks)
  repeat until (tickValue <> TickCount)
    ( start drawing fast!! here... )
```

That loop waits until the interrupt occurs, and then you should draw. Keep in mind that the top of the window should be drawn before the bottom (if possible) because the electron beam goes from top to bottom.

From: Tan
Re: Scrolling

What you could do is to use ScrollRect, that'll take care of most of the window that's scrolled. The new portion of the window contents that need to be redrawn can be clipped. As there's less to draw on the screen, it would be faster. Of course, another way is to cache a margin around the current visible area in an offscreen bitmap and blit it in when you scroll to the cached area. You'll need four caches for each of the four directions you can scroll in. I think Excel does something like that to speed the scrolling.

From: Moreno
Re: Scrolling

Here is some pseudo code for determining if you should draw a line in your apps window

```
theUpdateRgn := NewRgn; ( create a temporary region, NewRgn
returns a region HANDLE )
ScrollRect(windowRect_Minus_One_Line_at_Top_Or_Bottom, +_or_-_
_One_Line_Height,
0, theUpdateRgn);
( don't forget to subtrac the scrollbars from the window rect
)
trect:=theUpdateRgn^.rgnBBox;
SetRect(DrawingRect, ( you fill in the blanks it's not MY
program)left,top,right,bottom);
IF RectInRgn(DrawingRect, YourWindow^.visRgn) AND
RectInRgn(DrawingRect,
YourWindow^.clipRgn) THEN
BEGIN
( draw the top or bottom line if scrolling by on line )
END;
```

From: Inbox
Re: TEdit

Does anyone know of a good way to replace (or designate) the text in a Text Edit record? I did the following, and it didn't quite work... vat sheesh.. make that VAR

```
myTeh : TEHandle;
myStrHandle : StringHandle;
myStr : String[20];
begin
myStr := ' 20 letters go here ... ';
myStrHandle := NewString(myStr);
myTeh^.hText := Handle(myStrHandle);
TECaltText(myTeh);
end;
```

What happens is that a garbage char gets displayed along with the first 17 or so characters of myStr; And the ASCII value of the char is 20, and so is the length of the string. So, I think that the type coercion from StringHandle to Handle makes it point to the length of the string and not to the contents. I don't know what to do with this, because I'm using LSP 2.0. If this was in C, no problem. But I don't know how to do this in Pascal. So, I FOUGHT THE ROM AND THE ROM WON!!!! <a sob is heard here> ANY suggestions are greatly appreciated.

From: Gurglekak
Re: TEdit

Just in case you've not got an answer yet, the garbage

character is the Pascal length byte. NewString passes you back a StringHandle, which points to a chunk of mem headed by the length byte, which is followed by the letters of the string. In order to do what you are doing while maintaining your code, it might be best to do

```
BlockMove(@myStrHandle^[1], @myStrHandle^[0],
length(myStr));
SetHandleSize(Handle(myStrHandle)), GetHandleSize(
Handle(myStrHandle))-1);
```

before installing the handle and TECaltText'ing. However, I think you should use

```
TEInsert(@myStr[1], length(myStr), myTeh);
```

which avoids the StringHandle muddle altogether. TEInsert will take care of space for the text, if that's what you're worried about.

From: Gurglekak
Re: TEdit

Ah. Forgot to mention — the TECaltText becomes unnecessary if you use the TEInsert method.

From: Inbox
Re: TEdit

I thought that this was the garbage char. Thank you very much for your suggestion. However, I don't think that TEInsert is the way to go because I have to clear the text that was already there. Anyway, I'm using this scheme in a spreadsheet program I'm writing (and, no, I don't plan to put Microsoft out of business) for input to and from cells. The new text has to be shown when a new cell is selected, and that's where the garbage char problem comes in. Is there a better way to do cell input?

From: Pdyson
Re: Shutdown under Multifinder

A good Multifinder-friendly app quits gracefully when the user gives the Restart or Shutdown command from the Finder's Special menu. But what is the secret of writing an app to receive and obey this message from the Finder? I downloaded the TESample.C from Apple DTS, modified it for LSC 3.0 and Lo! it simply refuses to die when the Finder calls for a restart.

The app has 4 mstr resources (100-103) and (since I simply cribbed it from DTS) should be correct in the way it dispatches events. I am mystified.

From: MacDTS
Re: Shutdown under Multifinder

I would guess the "Quit" item in the File menu has more than what you're specifying in 'mstr'. Look for some spaces after the Quit string. I've heard about this problem before with LSC.

From: Rapler
Re: GrowZone, cbNeeded

I've got a GrowZone proc that seemed to work fine, except

that, occasionally, I get a negative number for cbNeeded. I'm using LSP 2.01, and my test program is allocating \$9000 bytes. The value of cbNeeded is FFFF9008!! Is this an OS bug, a weirdness with LSP's Zone, or something unheard of??.

I don't want to add a special negative number handler without knowing why, and I want to put off going into Macsbug if someone already knows what's happening.

Any help would be appreciated.

From: Brad

Re: GrowZone, cbNeeded

How are you allocating the memory? If you are using \$9000 as a constant, try changing it to \$09000, which will force the Compiler to use a LONGINT. It might be sign extending the \$9000 (which is -28672 in decimal), in which case you would be asking for a HUGE block of memory. I think the memory manager treats memory requests as unsigned.

From: Nfong

Re: ioFLAttrib

I'm trying to set the protect bit in ioFLAttrib with PBSetCatInfo but can't seem to get it to work. It returns with no error but doesn't do anything. Anyone got any suggestions?

From: Inbox

Re: bundle bit

Does anyone know whether it's possible to set the bundle bit using ResEdit? I'm using LSC 3.0, and couldn't get the icon to be recognized.

From: Spud

Re: bundle bit

There was an old version of ResEdit that had the Changed and Bundle bits mixed up; you clicked on the Bundle checkbox and the Changed bit was set, and vice versa. The good ol' boys at Apple have fixed that little mishap since then, and I'm sure that the version here (version 1.2) has the fix.

From: Eau

Re: bundle bit

Yes, you can set the bundle bit with ResEdit. Do 'command-I' when a file is selected, and a window will appear, which has a checkbox for Bundle. To get the Finder to recognize the icon, you will probably have to move the file to a different folder. The Finder will not automatically pick up changed bundle bits. Of course, the file should also contain resources defining the icons.

When I've got an application, and I want the Finder to read the bundle information, I move it between the top-level and a subdirectory on my hard disk. This works pretty well. Hope this helps.

Volume 5, Number 10

From: Willcox

Re: Deleting a Folder

I am having trouble deleting a folder using the Low Level

File Manager routines. I have created a folder and put some temporary files in it. At the end of my program, I would like to delete this directory. I am trying to use PBHDelete. I first delete all the files in this folder. Then, I try to close the working directory with PBCloseWD since I opened it with PBOpenWD. Then I call PBHDelete. I get error -47, fBsyErr, File is busy. I have tried various parameters in PBHDelete (WDId, vRefnum, name of folder), but I still get -47. Also, when I do this with MultiFinder turned off, I can not delete the folder from the Finder, it too says the folder is busy. Help.

From: Adept

Re: Deleting a Folder

When doing any PBOpen calls, you have to be sure to set your ioWDProcID to 'ERIK', the magic 'longint', otherwise it will be undeletable by the finder (until reboot). You might be calling PBClose with something != PBOpen and != to 'ERIK'..

From: Mikec

Re: Setting current folder

Bryce, I am excited to see your message #878 that creates new folders. I have been wanting to do this for an application for a long time. But I am just starting to program and have little idea how to fit your listed code into a procedure that will create the folder. Can you help me by listing a more complete procedure or uploading a text file? I use LSP and would be delighted to hear from you.

From: Tony

Re: Setting current folder

Mike, Below is the list of a function call CreateFolder. I believe this will do what you want. It takes as parameter the name of the folder and the volume reference of where to put the folder. In return, it will pass back an errorcode and the FolderID. The folderId is in fact the Directory Id and can be used as a vRefNum (volume reference no) in any of your File I/O calls.

```
Function CreateFolder(FolderName:str255; vRefNum : integer;
varFolderId:longint):OSErr;
Var
  Blk:HPParamBlockRec;
  errorcode :OSErr;
begin
  with blk do
    begin
      ioCompletion := nil;
      ioNamePtr := @FolderName;
      ioVrefNum := vRefNum;
      ioDirId := 0;
    end;
    errorcode := PBDirCreate(@blk,false);
    FolderID := blk.ioDirId;
    CreateFolder := errorcode;
  end;
```

Hope this helps.

From: Shaper

Re: List Manager problem...

Does anyone have any source code for the List Manager

routines? I got the source to the LDEF's in the Think Pascal library on CompuServe and tried to use that source but it's kind of strange because he used custom LDEFs with variable column numbers. I want to use the LDEF that comes in the system file to make a list of people's names, and I can get the list to appear but the only way to scroll it is to click on someone's name and then drag downward to scroll down - the arrow buttons don't work for some reason. Also, it doesn't always hilite the cell that you click on, so you have to keep clicking on different parts of the list until one does hilite before you can scroll up or down...

From: Jumpcut
Re: List Manager problem...

If you don't specify a LDEF, the List Manager will use the standard system LDEF. You can then set the list's selfFlags to make sure things get selected properly. Check around page 270 of IMIV. I was having similar problems after I changed the font and style of the window the list was in. First thing to check - make sure your window is a noGrowDocProc, or else the grow box (whether you draw one or not) will keep mouseDown events from scrolling the list down. I'm sure you're using LClick, so I won't bother to mention that. My only other suggestion is to avoid the obviously stupid (unlike me) and remember to change the font & face before creating the list.

Volume 5, Number 11

From: Jmoreno
Re: Dispatch Table

I'm trying to gain some experience with C by translating a program I have from Pascal to C. I don't want to do a line by line translation, and I noticed that I could handle menu events using a dispatch table to functions instead of a bunch of switch statements. My problem is that I can not get the dispatch table to compile under Think C 3.0. Could somebody please post an example of one?

From: Nolsy
Re: Dispatch Table

One way to code a dispatch table in C is to create an array of pointers to functions. That way you can provide your own indexing techniques and bypass multiple depths of switch() statements. This does have some limitations though: since all the calls are made through a single statement, ensuring that each function gets the correct type/number of parameters can be a headache (typecasting unsigned longs is probably your best bet). Secondly, heaven help you if you try to jump with an invalid index value...you'll discover the delights of the BRND (Branch to random address) opcode! Anyway...here's a little source to help you on your way:

```
typedef void (*FuncPtr)();  
FuncPtr func_list[] = { my_func1, my_func2, my_func3 };  
...  
index_val = 2; /* whichever value you want to jump to  
(*func_list[index_val])(parameters...);  
...  
Eh voila! Instant jump table.
```

From: Gpoole
Re: INITs

This is a fairly open-ended question but we would appreciate any help we can get on this matter. The subject is INIT's, or more specifically, how to get along with them. We are developing a program for commercial release and all seems to work fine as long as we are using just Finder or MultiFinder, no bombs, errors, weird happenings, quitting unexpectedly, etc. As soon as we have some INITs in the system though, all kinds of weird things start happening. With that said, are there any general guidelines we as developers should follow in coding to peacefully co-exist with INITs? Are there any known defensive programming techniques? In case it helps, some the INITs we have are: DialogKeys (CE Software), SuperLaserSpool, PowerMenus, Capture, CanOpener, and Suitcase. Thanks for any help you can give.

From: Gurglekat
Re: INITs

The only thing I can think of for general defensive programming against INITs is to assume that ALL traps move memory. Patched traps can, of course, move memory, and INITs patch traps... I don't remember where I heard this; maybe from one of Knaster's books.

From: Rguerra
Re: INITs

It's probably SAFEST to assume that ANY trap can move memory. While a patched trap's code in and of itself doesn't absolutely HAVE to move memory, there are no guarantees. If you're going to patch traps, one should avoid appending code that moves memory in a trap that is NOT supposed to do so. But who knows what goes on so ... Let's all be careful out there!

From: Gpoole
Re: INITs

Thanks for the replies to my INIT question, especially the one from DTS. I also asked them this same question and the reply was that there is nothing that can be done for defense. The trenches seem to know best.

From: Brad
Re: INITs

If your program doesn't get along with the INIT's you mentioned in your message, then either you have a knack for finding bugs in other people's code, or your program is doing something very wrong. I work on a very large commercial product, and I run with dozens of INITs, MultiFinder, EtherTalk, etc, and darned near all of the crashes I see are either bugs in my code or bugs (occasionally) in the system. Some INITs out there are downright dangerous and shouldn't be distributed, but many of those you mentioned could be considered standard equipment on many machines.

From: Brad

Re: INITs

It's ridiculous to program as if any trap can move memory. If someone patches a trap that is documented as NOT moving memory, and, as a result of something in the patch, memory may be moved, then the patch has a bug and shouldn't be used. Defensive programming is important; check your error codes, preflight, don't assume, etc - but there's no need to get carried away. Relocatable blocks don't relocate unless provoked, and they only get purged if they're purgeable and memory is tight. (But come System 7, and who knows...)

From: Gurglekat**Re: INITs**

It's not the tough to program as if any trap can move memory. All you really have to be careful of are WITH statements, keeping handles half-dereferenced across traps, and passing dereferenced elements of records in handles as VAR parameters. HLock and HUnlock aren't that hard to use.

From: Essam**Re: Simple INIT**

There was a beautiful example of an INIT that stays around (by patching GetResource and ExitToShell to beep when an application is launched or quit, found in MacTutor June 1989. Unfortunately, this INIT was in Forth. Does anyone have a LSP Pascal version of this code?

From: Inbox**Re: LSP I/O procs**

It seems that LSP's built-in I/O procs don't work properly with HFS. My app uses info passed by SFGGetFile in the Open(myFileVar,name) call but I can't access files that are in a directory other than the one the app is in. What should I do? Do I need to use PBGetCatInfo or something of this sort?!

From: Slegel**Re: LSP I/O procs**

If you're getting back from a standard file call, do a SetVol(NIL, reply.vRefNum) before calling the library routines.

From: Unclejim**Re: Disk Based Scrolling**

Help! I am still struggling with getting some disk based scrolling to work. I need to scroll around in 200-300K files and only keep 40-50K in memory at one time. (Scroll Up & down etc). I can't seem to keep my scroll bars moving smoothly as I swap buffers and use TSESetText etc. If anyone has even the simplest example of this disk based scroll in any language it would be a terrific help. Editing is not an issue at this stage just scroll up and down and keep track of the buffers. I thought it would be easy; it probably is, but I'm not having much luck so far. Thanks Jim

From: Jmoreno**Re: arrays as files**

Here is some pascal source for finding you were launched

with a file

```
CountAppFiles(AppMessage, NumAppFiles);
IF NumAppFiles < 0 THEN ( the user has opened documents from
the finder )
that would be something like
CountAppFiles(AppMessage,&NumAppFiles);
if NumAppFile != 0 /* open the file */
```

Volume 5, Number 12

From: Willcox**Re: Print Manager Error**

I solved my printing problem, but I do not fully understand why it now works. Maybe someone out there can explain. First, the problem was that sometime when printing (using a Mac IIcx only), the system would sometimes forget that a certain font was a LaserFont, and promptly started creating a bitmap for it. However, the screen font information was also lost, so this process eventually terminated in an error.

I tracked the problem down to a single call to "UseResFile" from which I had to load a specific private resource. I fixed the problem by modifying the code to:

```
temp:=CurResFile;
UseResFile(theFileNumber);
theResource:=GetResource(asdfjasldf);
UseResFile(temp);
```

Now, why does this work??? I have not found a warning in Inside Macintosh, and the error only occurs on my Mac IIcx, not on my Mac Plus.

From: Jmoreno**Re: Print Manager Error**

What was happening is the call to UseResFile was removing the System from the list of open files to search for a resource. And that font was no longer found. The system adds each opened resource to the end of a list and then when you try to load a resource it starts at the end and backtracks looking through all of the open resource forks for the requested resource. So when you open a resource it gets a place in line, when you call UseResFile it gets put at the beginning and is set to be the FIRST resource to search (everything else is then gone from the resource list). Hope this helps you understand what is going on. BTW if you haven't already I STRONGLY recommend that you buy Inside Mac Vol 1 to 5.

From: Inbox**Re: Var declaration**

Sheeshh.... Here I go again..

Awright, I'm trying to create an array of the following size after VAR in my globals unit: myarray : array[1..7,1..300] of string[20]; but LSP says that it's "unable to create blah..blah..blah on this level". Now what is a body to do if using dynamic memory allocation means changing the whole program?? Is there some kind of a call I can make to perhaps increase the size of the applications stack beforehand?! But the globals' unit gets compiled first so the instruction won't get executed at that point...

Thanks for listening, _inBox_ (this should read _in-

Trouble_)

From: Jmoreno
Re: Var declaration

In the run options part of Think Pascal (formerly LSP) there is an option for setting both the stack (which is what you want) and the heap.

From: Tomt
Re: Var declaration

Dumb question but how in Pascal do I access the stuff in a handle. My problem is that I need to make an array of integers of size, say, 500 by 500. Now I tried array type of that size, without making any instances of it, but LSP wouldn't let do that. So if I make a handle to the appropriately sized chunk of memory, how do I access the i,j the element. I'm sure its trivial, but I haven't been able to figure it out. Thanks in advance for any help

From: Inbox
Re: Var declaration

Awright.. Methinks I got it.. Here's the 500 by 500....

```
TYPE
myarray = array[1..500,1..500] of integer;
myarrayptr = ^myarray;
myarrayhandle = ^myarrayptr;
VAR
THEarray : myarrayhandle;
BEGIN
THEarray := myarrayhandle(NewHandle(SizeOf(myarray)));
That's it. To access an element, do
myint := THEarray^[i,j]; where i and j are integers, etc...
Set an element by THEarray^[i,j] := myint;
Hope this helps. _inbox_
```

From: Jimm
Re: Var declaration

On page 447 of LSP 2.0 manual it says that data structures cannot exceed 32766 bytes. Your data structure is 22*7*300 or >42K.

From: Dhands
Re: Var declaration

The previous VAR declaration will work as stated, since the data structure allocated at compile time is only a 4 byte handle, not the >40k array. The array is later allocated dynamically at run-time.

For example:

```
VAR
good : BigHandle; {4 byte handle to huge, >32k, data}
bad : ARRAY[1..40000] char; {>32k data won't fit here}
```

The difference is that "good" has its data on the heap (via NewHandle) and is only limited in size by the available memory. Where as "bad" is pre-allocated space in global or local areas of the heap which is limited to 32k.

From: Shaper
Re: Printing Samples?

One more question, does anyone have any Pascal (preferably LSP) source code for printing to an ImageWriter - text only, no graphics necessary, I tried to use PrCtlCall() or whatever it's called, but I don't know how to set up a ptr to text, and also how do you set the HiWord and LoWord of a longint? I know you can GET it by doing aLongInt:=HiWord(bLongInt) or aLongInt:=LoWord(bLongInt) but you can't do HiWord(bLongInt):=aLongInt...

From: Dhands
Re: Printing Samples?

Here's a quick summary of draft printing using the print manager's low level calls. The low level calls should not be used for "real" printing, but they are great for quick-n-dirty printing.

```
PrDrvOpen;                                {open print driver}
PrCtlCall(iPrDevCtl,1PrReset,0,0);         {reset/init printer}
PrCtlCall(iPrIOCtl,1LONGINT(@textBuf),count,0); {send some
text}
PrCtlCall(iPrDevCtl,1PrLFSixth,0,0);       {advance printer pa-
per}
PrDrvClose;                                {close print driver}
VAR
textBuf : PACKED ARRAY [1..256] OF Char;
str      :      Str255;
```

Since you must explicitly advance the paper you have to print one line at a time. To print a Str255 for example:

```
count := Length(str) {number of bytes to copy}
FOR i := 1 TO count DO
textBuf[i] := str[i]; {note: don't copy over str[0] length
byte}
PrCtlCall(iPrIOCtl,1LONGINT(@textBuf),count,0);
PrCtlCall(iPrDevCtl,1PrLFSixth,0,0);
```

From: Jmoreno
Re: Printing Samples?

1a) Why don't you try using the regular printing process, it's no big deal to open the port then open a page print a page, close the page then open another page if needed....

1b) To get a pointer to the text in a Str255 use

TextPtr:=@YourString[1]; { really need to make sure you have 1+ chars}

2a) to set the Hi&LoWord of a longInt use:

```
bLongInt:LongInt;
theInt:=Integer;
BEGIN
theInt:=@bLongInt;
theInt:=12; { set the HiWord}
theInt:=Ptr(LongInt(theInt)+2);
theInt:=60; { set the LowWord}
END;
or Use a variable RECORD
```



Topical Index

The Best of MacTutor, Volume 5, 1989

Topic	Language	Title	Best of Journal Source			
			Page	No.	Pg.	Disk
AACK	Pascal	AACK Gives AppleTalk Shell	514	12	10	51
ADB	Basic, Pascal	ADB Demo in ZBasic 5.0 & LS Pascal 2.0	562	3	21	42
ADSP	HyperChat, Pascal	XCMD Corner: HyperAppleTalk, Part II	611	2	91	41
ADSP	Pascal, HyperTalk	AppleTalk DSP Part III	619	3	53	42
ADSP	Pascal, HyperTalk	XCMD Corner: HyperAppleTalk IV	643	4	54	43
Alarms	Basic	Time Retrieval And Storage Techniques	559	1	33	40
Aliasing	Pascal	Dispatch Aliasing	503	11	75	50
Apple Data Stream Protocol	HyperChat, Pascal	XCMD Corner: HyperAppleTalk, Part II	611	2	91	41
AppleTalk	C	AppleTalk Chatter Box	227	12	26	51
AppleTalk	Forth	AppleTalk Protocols	554	7	52	46
AppleTalk	HyperChat, Pascal	XCMD Corner: HyperAppleTalk, Part II	611	2	91	41
AppleTalk	HyperTalk, Pascal	HyperAppleTalk	591	1	71	40
AppleTalk	Pascal	AACK Gives AppleTalk Shell	514	12	10	51
AppleTalk	Pascal	Build a Network Painting Program	379	7	28	46
AppleTalk	Pascal, HyperTalk	AppleTalk DSP Part III	619	3	53	42
Arrays	Hypertalk, Pascal	HyperArrays	662	6	104	45
ATP	HyperTalk, Pascal	HyperAppleTalk	591	1	71	40
Bezier Curves	C	Bezier Curve Ahead!	49	1	39	40
BitMap	C, Assembly, General	Dot Matrix Printer Driver	149	10	22	49
BitMaps	Pascal, Hypertalk	XCMD Corner: Window Scrollbars	626	3	62	42
C++	C	Apple's C++	222	11	60	50
C++	C	C++ Overview	138	9	42	
C++	C	C++ Sample Application	239	12	84	51
Catalog	HyperTalk, C	XCMD Corner: Catalog XCMD	679	11	72	50
CDEF	C	Alternate Scroll Bar CDEF	213	11	48	50
CDEF	Pascal	Designer CDEFs	309	4	85	43
CDEF	Pascal	Finder Icon Controls	246	1	8	40
CDEF	Pascal	Have Your Window Do The Splits	354	6	66	45
CDEF	Pascal	New CDEF Messages	418	8	36	47
CDEV	C	World CDEV Investigated	146	9	88	48
CDEV	Forth	System CDEV	545	5	51	44
Chatter Box	C	AppleTalk Chatter Box	227	12	26	51
Chooser	C, Assembly, General	Dot Matrix Printer Driver	149	10	22	49
Clipboard	C	Extend Your Favorite Editor	122	7	15	46
Code Optimization	Assembly, Pascal	Assembly Language for the Rest of Us	38	12	56	51
Code Segments	C	Segmenting DA's	93	5	36	44
Color	Pascal, General	Understanding Color Icons	492	10	10	49
Commando	Pascal General	MPW Tools with TML Pascal II	319	5	22	44
Communications	Pascal, HyperTalk	AppleTalk DSP Part III	619	3	53	42
Communications	Pascal, HyperTalk	XCMD Corner: HyperAppleTalk IV	643	4	54	43
Compilers	General	Compiler Comparisons	702	8	94	47
Control Definitions	Pascal	Finder Icon Controls	246	1	8	40
Controls	C	Alternate Scroll Bar CDEF	213	11	48	50
Controls	C	Inside Controls	190	11	22	50
Controls	Pascal	Designer CDEFs	309	4	85	43
Controls	Pascal	MIDI User Interface	336	5	73	44
CopyBits	Pascal, Hypertalk	XCMD Corner: Window Scrollbars	626	3	62	42
Copying Files	C	XCMD Corner: CopyFile	677	10	55	49
CopyMask	Pascal	Spray Can and Paint Bucket Tools	366	6	80	45

Topic	Language	Title	Best of Journal Source			
			Page	No.	Pg.	Disk
Cursor Animation	C	Lisa's Hourglass Is Back!	100	5	104	44
Cursor Animation	Pascal General	MPW Tools with TML Pascal II	319	5	22	44
Debugging	MPW	SADE Debugging	707	9	76	48
Desk Accessories	C	Extend Your Favorite Editor	122	7	15	46
Desk Accessories	C	Segmenting DA's	93	5	36	44
Desk Scap	C	Extend Your Favorite Editor	122	7	15	46
Device Manager	Assembly	Cerial (Serial) I/O	19	7	78	46
Dialog	Pascal	Security Patrol for Viruses	261	2	23	41
Dialog Filter	Assembly	Help! An Easy to use Help Function	2	1	82	40
Dialogs	Assembly	Help! An Easy to use Help Function	2	1	82	40
DITL	C	Three Doodats	142	9	82	48
Driver	Assembly	Real-Time Driver	31	10	101	49
Driver	C, Assembly, General	Dot Matrix Printer Driver	149	10	22	49
Editing	C	Extend Your Favorite Editor	122	7	15	46
Events	C	Event Simulator	54	1	100	40
Events	C	Three Doodats	142	9	82	48
File I/O	Pascal	Security Patrol for Viruses	261	2	23	41
Finder	Pascal	Dispatch Aliasing	503	11	75	50
Finder Controls	Pascal	Finder Icon Controls	246	1	8	40
FKEY	C	Event Simulator	54	1	100	40
FKEY	Pascal	Capture Color Icons	528	12	68	51
FKEY	Pascal	Capture Color Icons	528	12	68	51
Font Tool	Pascal	How To Write a Font Tool for MPW	289	3	101	42
Fonts	General	New Typographic Frontier	712	10	77	
Growing Windows	Pascal	Re-sizing with Window Corners	295	4	67	43
Help Function	Assembly	Help! An Easy to use Help Function	2	1	82	40
Help Function	Pascal, Basic	Help for HELP Software	583	6	23	45
HFS	HyperTalk, C	HFS Tree Climbing	671	8	51	47
Hierarchical File System	HyperTalk, C	HFS Tree Climbing	671	8	51	47
HyperAppleTalk	HyperChat, Pascal	XCMD Corner: HyperAppleTalk, Part II	611	2	91	41
HyperCard	HyperTalk, C	HFS Tree Climbing	671	8	51	47
HyperServer	HyperTalk	A Look Into Networking	681	12	74	51
HyperTalk	HyperTalk	A Look Into Networking	681	12	74	51
Icons	Pascal	Capture Color Icons	528	12	68	51
Icons	Pascal	Finder Icon Controls	246	1	8	40
Icons	Pascal, General	Understanding Color Icons	492	10	10	49
Indexing	C	Three Doodats	142	9	82	48
INIT	C	April Fool's INIT: Scrolling the Menu Bar!?	88	4	39	43
INIT	C	Change the Boot Disk Icon	98	5	48	44
INIT	C	Writing INITs in THINK C	174	10	83	49
INIT	Forth	INITs and Patches	550	6	29	45
INIT	Pascal	Writing INITs in Pascal	501	10	51	49
Installer	General	The Installer Explored	697	8	28	47
Keyboard	Basic, Pascal	ADB Demo in ZBasic 5.0 & LS Pascal 2.0	562	3	21	42
Launching	Pascal	Dispatch Aliasing	503	11	75	50
LDEF	Pascal	List of Controls LDEF	371	6	86	45
List Manager	C	How to Write a Spreadsheet in LSC	58	3	84	42
MacApp	Pascal	A Tale of Two Quadratic Plotters, Part I	432	8	56	47
MacApp	Pascal	A Tale of Two Quadratic Plotters, Part I	432	8	56	47
MacApp	Pascal	A Tale of Two Quadratic Plotters, Part II	454	9	18	48
MacApp	Pascal	A Tale of Two Quadratic Plotters, Part II	454	9	18	48
MacApp	Pascal	Introduction to MacApp	409	8	10	47
MDEF	Pascal	Finder Icon Controls	246	1	8	40

Topic	Language	Title	Best of Journal Source			
			Page	No.	Pg.	Disk
MDS	Assembly	MDS->MPW	17	6	20	
Menu Bar	Pascal Hypertalk	Menu Bar XCMD	651	5	58	44
Menu Definitions	Pascal	Finder Icon Controls	246	1	8	40
Menus	Basic	Pop Up Menus in True Basic	575	4	47	43
Menus	C	April Fool's INIT: Scrolling the Menu Bar!?	88	4	39	43
Menus	Pascal	A Window Menu	400	7	93	46
Meters	C	Inside Controls	190	11	22	50
MIDI	C	Apple's MIDI Manager	181	11	10	50
MIDI	Pascal	MIDI User Interface	336	5	73	44
Modeless Dialogs	Pascal	Modeless Search and Replace	344	5	92	44
MPW	Assembly	MDS->MPW	17	6	20	
MPW	MPW	SADE Debugging	707	9	76	48
MPW	Pascal	How To Write a Font Tool for MPW	289	3	101	42
MPW Tools	Pascal General	MPW Tools with TML Pascal II	319	5	22	44
Munger	Pascal	Modeless Search and Replace	344	5	92	44
NBP	HyperChat, Pascal	XCMD Corner: HyperAppleTalk, Part II	611	2	91	41
NBP	HyperTalk, Pascal	HyperAppleTalk	591	1	71	40
Networking	C	AppleTalk Chatter Box	227	12	26	51
Networking	HyperTalk	A Look Into Networking	681	12	74	51
Networking	HyperTalk, Pascal	HyperAppleTalk	591	1	71	40
Networking	Pascal	AACK Gives AppleTalk Shell	514	12	10	51
Networking	Pascal, HyperTalk	AppleTalk DSP Part III	619	3	53	42
Networking	Pascal, HyperTalk	XCMD Corner: HyperAppleTalk IV	643	4	54	43
Networks	HyperChat, Pascal	XCMD Corner: HyperAppleTalk, Part II	611	2	91	41
Networks	Pascal	Build a Network Painting Program	379	7	28	46
Networks	Pascal	Build a Network Painting Program	379	7	28	46
Notification Manager	Pascal	The Notification Manager	330	5	41	44
Object Oriented Programming	C	Apple's C++	222	11	60	50
Object Oriented Programming	C	C++ Overview	138	9	42	
Object Oriented Programming	C	C++ Sample Application	239	12	84	51
Object Oriented Programming	C	Pseudo Objects	129	8	38	47
Object Oriented Programming	Pascal	Introduction to MacApp	409	8	10	47
PACK	C, Assembly, General	Dot Matrix Printer Driver	149	10	22	49
Paint Bucket	Pascal	Spray Can and Paint Bucket Tools	366	6	80	45
Palletes	C	All About Scrolling Windows	69	4	11	43
Parameter RAM	Basic	Time Retrieval And Storage Techniques	559	1	33	40
PDEF	C, Assembly, General	Dot Matrix Printer Driver	149	10	22	49
PICT	Assembly	Convert PICTs to Regions	11	6	10	45
Pop up Menus	Pascal	MIDI User Interface	336	5	73	44
PopUp Menus	Basic	Pop Up Menus in True Basic	575	4	47	43
Popup Menus	Pascal	List of Controls LDEF	371	6	86	45
PostScript	C	Bezier Curve Ahead!	49	1	39	40
PREC	C, Assembly, General	Dot Matrix Printer Driver	149	10	22	49
PREC	Pascal	A Look at the PREC Resource	420	8	80	47
Printing	C, Assembly, General	Dot Matrix Printer Driver	149	10	22	49
Printing	General	20 Steps to Printing Incompatibility	687	6	58	45
Printing	Pascal	A Look at the PREC Resource	420	8	80	47
Protocols	Pascal, HyperTalk	AppleTalk DSP Part III	619	3	53	42
Record Definitions	Forth	Record Definitions	539	2	14	41
Regions	Assembly	Convert PICTs to Regions	11	6	10	45
Rulers	C	All About Scrolling Windows	69	4	11	43
SADE	MPW	SADE Debugging	707	9	76	48
Scrap Manager	C	Extend Your Favorite Editor	122	7	15	46

Topic	Language	Title	Best of Journal Source			
			Page	No.	Pg.	Disk
Scroll bar	C	Alternate Scroll Bar CDEF	213	11	48	50
Scrollbars	C	All About Scrolling Windows	69	4	11	43
Scrollbars	Pascal, Hypertalk	XCMD Corner: Window Scrollbars	626	3	62	42
Sectors	C, Hypertalk	Sector Dumps	667	7	60	46
SeedFill	Pascal	Spray Can and Paint Bucket Tools	366	6	80	45
Serial I/O	Assembly	Cerial (Serial) I/O	19	7	78	46
snd	Pascal	'snd ' Zounz Great!	475	9	56	48
Sound	Basic	Sounding Off with MS QuickBasic	578	5	68	44
Sound	C	Apple's MIDI Manager	181	11	10	50
Sound	Pascal	'snd ' Zounz Great!	475	9	56	48
Splitbar	Pascal	Have Your Window Do The Splits	354	6	66	45
Spray Can	Pascal	Spray Can and Paint Bucket Tools	366	6	80	45
Spreadsheet	C	How to Write a Spreadsheet in LSC	58	3	84	42
Subscripts	C	Sub and Superscripting with TE	110	6	42	45
Superscripts	C	Sub and Superscripting with TE	110	6	42	45
SysEnviorns	Basic, Pascal	ADB Demo in ZBasic 5.0 & LS Pascal 2.0	562	3	21	42
SysEnviorns	Forth	System CDEV	545	5	51	44
System 7.0	General	Developer Conference Report	693	7	3	
System Clock	Basic	Time Retrieval And Storage Techniques	559	1	33	40
System Folder	C	The Blessed Folder	674	9	52	48
System Queues	Forth	System Queues	533	1	49	40
Terminal Emulator	Assembly	Cerial (Serial) I/O	19	7	78	46
TextEdit	C	Sub and Superscripting with TE	110	6	42	45
TickCount	Basic	Time Retrieval And Storage Techniques	559	1	33	40
Trap Patches	Forth	INITs and Patches	550	6	29	45
Trap Patching	C	Sub and Superscripting with TE	110	6	42	45
Trap Patching	C	Writing INITs in THINK C	174	10	83	49
Trap Patching	Pascal	Writing INITs in Pascal	501	10	51	49
Tutorial	Assembly, Pascal	Assembly Language for the Rest of Us	38	12	56	51
Virus	Pascal	Security Patrol for Viruses	261	2	23	41
WDEF	Pascal	Re-sizing with Window Corners	295	4	67	43
Windows	C	All About Scrolling Windows	69	4	11	43
Windows	C	Window Structures	107	6	35	45
Windows	HyperChat, Pascal	A Real Window XCMD	599	2	74	41
Windows	Pascal	A Window Menu	400	7	93	46
Windows	Pascal	Re-sizing with Window Corners	295	4	67	43
Windows	Pascal Hypertalk	Menu Bar XCMD	651	5	58	44
XCMD	C	The Blessed Folder	674	9	52	48
XCMD	C	XCMDs in Think C	653	5	84	44
XCMD	C, Hypertalk	XCMD Libraries	659	6	101	45
XCMD	HyperChat, Pascal	A Real Window XCMD	599	2	74	41
XCMD	HyperTalk, C	HFS Tree Climbing	671	8	51	47
XCMD	HyperTalk, C	XCMD Corner: Catalog XCMD	679	11	72	50
XCMD	HyperTalk, Pascal	HyperAppleTalk	591	1	71	40
XCMD	Pascal Hypertalk	Menu Bar XCMD	651	5	58	44
XCMD	Pascal, Hypertalk	XCMD Corner: Window Scrollbars	626	3	62	42
ZBasic	Basic	Useful Routines For ZBasic	587	7	104	46

Author's Index

The Best of MacTutor, Volume 5, 1989

Author	Language	Title	Best of Issue Source		
			Page	No.Pg.	Disk
Alviani, Frank	General	New Typographic Frontier	712	10 77	
Austin, Kirk	Pascal	MIDI User Interface	336	5 73	44
Butcher, Ron	Basic	Sounding Off with MS QuickBasic	578	5 68	44
Chase, Kirk	Pascal	Designer CDEFs	309	4 85	43
Chase, Kirk	Pascal	Have Your Window Do The Splits	354	6 66	45
Chase, Kirk	Pascal	'snd ' Zounz Great!	475	9 56	48
Cohn, Ted	Assembly	Convert PICTs to Regions	11	6 10	45
Colwell, Alexander S.	C	Alternate Scroll Bar CDEF	213	11 48	50
Combret, Jean de	Pascal	Finder Icon Controls	246	1 8	40
Dunham, David	C	Change the Boot Disk Icon	98	5 48	44
Earle, Ken	C	Inside Controls	190	11 22	50
Gordon, Bob	C	Window Structures	107	6 35	45
Hafernik, Rob	C	AppleTalk Chatter Box	227	12 26	51
Henriquez, Frank	Assembly	Cerial (Serial) I/O	19	7 78	46
Hoddie, J. Peter	C	Writing INITs in THINK C	174	10 83	49
Holder, John	Assembly	Help! An Easy to use Help Function	2	1 82	40
Horton, Earle	C, Assembly, General	Dot Matrix Printer Driver	149	10 22	49
Horton, Roger	Pascal	Dispatch Aliasing	503	11 75	50
Kane, Jeffrey B., MD	Assembly, Pascal	Assembly Language for the Rest of Us	38	12 56	51
Kelly, Dave	Pascal	A Look at the PREC Resource	420	8 80	47
Kelly, Dave	Basic	Time Retrieval And Storage Techniques	559	1 33	40
Kelly, Dave	Basic, Pascal	ADB Demo in ZBasic 5.0 & LS Pascal 2.0	562	3 21	42
Kelly, Dave	Basic, Pascal	Help for HELP Software	583	6 23	45
Kelly, Dave	Basic	Pop Up Menus in True Basic	575	4 47	43
Kelly, Dave	Basic	Useful Routines For ZBasic	587	7 104	46
Kiene, Steve	Pascal	Writing INITs in Pascal	501	10 51	49
Koscheka, Donald	HyperTalk, Pascal	HyperAppleTalk	591	1 71	40
Koscheka, Donald	HyperChat, Pascal	XCMD Corner: HyperAppleTalk, Part II	611	2 91	41
Koscheka, Donald	Pascal, HyperTalk	AppleTalk DSP Part III	619	3 53	42
Koscheka, Donald	Pascal, HyperTalk	XCMD Corner: HyperAppleTalk IV	643	4 54	43
Koscheka, Donald	C	XCMDs in Think C	653	5 84	44
Koscheka, Donald	C, HyperTalk	XCMD Libraries	659	6 101	45
Koscheka, Donald	C, HyperTalk	Sector Dumps	667	7 60	46
Koscheka, Donald	HyperTalk, C	HFS Tree Climbing	671	8 51	47
Koscheka, Donald	C	The Blessed Folder	674	9 52	48
Koscheka, Donald	C	CopyFile	677	10 55	49
Koscheka, Donald	HyperTalk, C	XCMD Corner: Catalog XCMD	679	11 72	50
Koscheka, Donald	HyperTalk	A Look Into Networking	681	12 74	51
Langowski, Jörg	General	Compiler Comparisons	702	8 94	47
Langowski, Jörg	C	C++ Overview	138	9 42	
Langowski, Jörg	C	Apple's C++	222	11 60	50
Langowski, Jörg	C	C++ Sample Application	239	12 84	51
Langowski, Jörg	Forth	AppleTalk Protocols	554	7 52	46
Langowski, Jörg	Forth	Record Definitions	539	2 14	41
Langowski, Jörg	Forth	System CDEV	545	5 51	44
Langowski, Jörg	Forth	System Queues	533	1 49	40
Langowski, Jörg	Forth	INITs and Patches	550	6 29	45
Leonard, Randy	Pascal	How To Write a Font Tool for MPW	289	3 101	42

Author	Language	Title	Bestof Page	Issue No.Pg.	Source Disk
Leonard, Randy	Pascal General	MPW Tools with TML Pascal II	319	5 22	44
Magnuson, Rod	Pascal	Build a Network Painting Program	379	7 28	46
Mandel, Jeff E., MD MS	Assembly	Real-Time Driver	31	10 101	49
McBride, Mark	Pascal	Modeless Search and Replace	344	5 92	44
McKenzie, Robert	C	Sub and Superscripting with TE	110	6 42	45
McMath, Chuck	Pascal	A Tale of Two Quadratic Plotters, Part I	432	8 56	47
McNamara, Joel	C	Extend Your Favorite Editor	122	7 15	46
Minow, Martin	C	World CDEV Investigated	146	9 88	48
Morton, Mike	C	Lisa's Hourglass Is Back!	100	5 104	44
Nairn, John A.	C	All About Scrolling Windows	69	4 11	43
Nath, Ajay	Pascal	AACK Gives AppleTalk Shell	514	12 10	51
Nelson, Carl	Pascal	A Tale of Two Quadratic Plotters, Part II	454	9 18	48
Neuse, Lee A.	C	Three Doodats	142	9 82	48
Parichan, Kevin	Pascal	Spray Can and Paint Bucket Tools	366	6 80	45
Pillera, Joe	MPW	SADE Debugging	707	9 76	48
Plamondon, James	Pascal	List of Controls LDEF	371	6 86	45
Plamondon, James	Pascal	New CDEF Messages	418	8 36	47
Saxton, Tom	C	Segmenting DA's	93	5 36	44
Scanlin, Mike	C	April Fool's INIT: Scrolling the Menu Bar!?	88	4 39	43
Seaquist, Steven	Pascal	Security Patrol for Viruses	261	2 23	41
Sheets, Steven	Pascal	The Notification Manager	330	5 41	44
Sheets, Steven	General	The Installer Explored	697	8 28	47
Sheets, Steven & Patricia	Pascal, General	Understanding Color Icons	492	10 10	49
Sheets, Steven & Patricia	Pascal	Capture Color Icons	528	12 68	51
Smith, David E.	Pascal	Introduction to MacApp	409	8 10	47
Smith, David E.	General	Developer Conference Report	693	7 3	
Smith, David W.	C	Bezier Curve Ahead!	49	1 39	40
Snyder, Matthew J.	C	Event Simulator	54	1 100	40
Stauder, Fred	Pascal Hypertalk	Menu Bar XCMD	651	5 58	44
Story, Clifford	Pascal	Re-sizing with Window Corners	295	4 67	43
Story, Clifford	Pascal	A Window Menu	400	7 93	46
Treister, Adam	C	Pseudo Objects	129	8 38	47
Veca, Don	C	Apple's MIDI Manager	181	11 10	50
Vore, Scott	Hypertalk, Pascal	HyperArrays	662	6 104	45
Waters, Bryan	C	How to Write a Spreadsheet in LSC	58	3 84	42
West, Joel	General	20 Steps to Printing Incompatibility	687	6 58	45
Weston, Dan	Assembly	MDS->MPW	17	6 20	
Zuffoletto, Joe	HyperChat, Pascal	A Real Window XCMD	599	2 74	41
Zuffoletto, Joe	Pascal, Hypertalk	XCMD Corner: Window Scrollbars	626	3 62	42

Articles NOT Included in The Best of MacTutor, Volume 5, 1989

Author	Title	Issue Source	
		No.Pg.	Disk
Wootton, Alan	Trace Lightspeed Pascal	1 56	40
Stauder, Fred	ScriptTips II	1 68	
Clark, Richard	Keyboard Repair	1 97	40
Kelly, Dave	Zedcor Compiler 5.0	2 9	41
Odom, Gary	Programmer's Forum	2 63	41
Chandler, Dick	C Workshop	2 66	41
Sweet, Jerry	Call the Mac ToolBox from Ada	3 13	
Langowski, Jörg	Porting Fortran to Mach 2	3 40	42
Stauder, Fred	String Width XCMD	3 51	42
Stauder, Fred	HyperChat Award	4 52	43
Love, John A., III	Icons in DA Menus	4 100	43
Kelly, Dave	User Log and Programming Tools	5 63	44
Lange, Jean Pascal J.	Animated Towers of Hanoi	6 95	45
Snively, Paul	NeXT Evolution	7 25	
Alviani, Frank	Apple's IAC in System 7	8 22	
Stauder, Fred	Bits And Pieces	8 49	
McDermott, Pat	Ada Compiler Review	9 12	
Kelly, Dave	AppMaker and Prototyper	9 48	
McBride, Mark	FaceIt Saves Fortran	9 92	48
Muys-Vasovic, Jean-Denis	Back to the Future: OOP & MacApp	9 102	
Kelly, Dave	After Dark and S.P.A.M.M	10 91	49
Langowski, Jörg	Objects in C; NEON; Letters	10 95	49
Kelly, Dave	Dialoger™ Helps HyperCard	11 68	
Schlienger, Eric	Repairing the ADB	11 88	
Lange, Jean-Pascal J.	OOP in Lisp	12 41	51
Kelly, Dave	ScriptEdit and WindowTalk	12 74	

\$34.95

